Aspects of Realism: Golf Simulation

Details:
Computing Individual Project
Matthew Winchester (97501464) <Winchester@imtoosexy.com>
BSc (Hons) Software Engineering
Academic Year:
2000 - 2001
Acknowledgements:
Project Supervisor - Peter Aughton
Word Count:
14922 words

Contents

1. INTRODUCTION 5

2. PROJECT DEFINITION 7

2.1 OVERALL AIM 7
2.2 OBJECTIVES 7

3. PROJECT PLAN 8

3.1 WORK BREAKDOWN 8
3.2 TIME ESTIMATION 10
3.3 MILESTONES 11
3.4 ACTIVITY SEQUENCE 12

4. RESEARCH: MAP CONTOUR 14

4.1 MATHEMATICAL REPRESENTATION OF CURVES 14
4.2 FREEFORM POLYGONAL MESH 15
4.3 STRUCTURED POLYGONAL MESH 17

5. RESEARCH: MAP TEXTURE 20

5.1 LIGHT SHADING 20
5.2 TEXTURE MAPPING 21
5.3 BUMP MAPPING 21
5.4 BI-DIRECTIONAL REFLECTANCE AND TEXTURE 22
5.5 SURFACE INTERACTION 23
5.6 DECISIONS 24

6. RESEARCH: BALL TRAVEL 25

6.1 EQUATIONS OF MOTION 25
6.2 GRAVITY 25
6.3 AIR AND WIND 26

6.4 Density, Temperature, Pressure & Humidity 27
6.5 Effects of spin in flight 27
6.6 Contact with ground 29
6.7 Rolling 30

7. RESEARCH: GOLF CLUBS 32

7.1 Types of Club 32
7.1.1 Wood 32
7.1.2 Iron 32
7.1.3 Putters 33
7.2 Effect on Ball 33
7.3 When to use which club 34

8. RESEARCH: 3D ENGINE 35

8.1 Basic Theory 35
8.2 The graphics pipeline 35
8.3 Major issues 36
8.3.1 Accuracy v Speed 36
8.3.2 Hidden Surfaces & Clipping 37
8.3.3 Shading / Texturing 38

9. RESEARCH: GOLF COURSE DESIGN 39

9.1 Greens 39
9.2 Bunkers 39
9.3 Teeing Grounds 40

10. RESEARCH: USER INTERFACE 41

10.1 Theory 41
10.2 Possible GUI design 41

11. REQUIREMENTS 43

R1 Ball Travel 43
R1.1 Ball Flight 43
R1.2 Ball Impact with Ground 44
R1.3 Ball Roll 44
R2 Golf Course 45
R2.1 Map Contour 45
R2.2 Map Texture 46
R3 Game 47
R3.1 Game Rules 47
R3.2 Golf Clubs 47
R4 Display 48
R4.1 3D Engine 48
R4.2 User Interface 48
R5 Objects 49
R6 Documentation 49

12. RISK 51

13. MAP MODELLING AND REPRESENTATION 52

13.1 MAP REPRESENTATION 52
13.2 MAP DISPLAY 52

14. BALL TRAVEL ALGORITHMS 58

14.1 DATA STRUCTURE 58
14.2 THE APPLICATION 59

15. DESIGN 60

15.1 CLASS AND OBJECT DIAGRAMS 60
15.2 STATE CHARTS 64
15.3 SEQUENCE DIAGRAMS 65

16. TEST PLAN 69

17. GAME IMPLEMENTATION 70

17.1 CODING STANDARDS 70
17.1.1 Naming Conventions 70
17.1.2 Requirements Tracing 70
17.1.3 Commenting 70
17.2 SCREENSHOTS 71
17.3 SKIPPED REQUIREMENTS 74
17.4 KNOWN BUGS 75
17.5 RESULTS FROM TEST PLAN 75

18. PROBLEMS AND CHANGES 76

18.1 R.U.P. 76
18.2 USER INTERFACE 76
18.3 SCHEDULING 76
18.4 REAL WORLD OBJECTS 77
18.5 BALL FLIGHT ANIMATION 77
18.6 BALL SHADOW 77
18.7 THE WORD COUNT 78

19. FUTURE WORK 79

19.1 FIXING THE LAG PROBLEM 79
19.2 REAL WORLD OBJECTS 79
19.3 MULTIPLE PLAYERS 79
19.4 NETWORK PLAY 79
19.5 MULTIPLE HOLES 79
19.6 LEVEL EDITOR 79

20. REFERENCES 80

APPENDIX A: PROJECT PROPOSAL 81

1.1 INTRODUCTION 81
1.2 AIMS AND OBJECTIVES 83
1.3 EXPECTED OUTCOMES / DELIVERABLES 83

1.4 METHODS 84
1.5 PROJECT PLAN 85
1.6 RESOURCE REQUIREMENTS 85
1.7 KEYWORDS 86
APPENDIX B: USER GUIDE 87
1. RUNNING THE SIMULATION 87
2. PLAYING THE SIMULATION 87
3. FEATURES OF THE SIMULATION 89
APPENDIX C: SOURCE CODE 90

1. Introduction

As the title suggests I am planning to create a simulation of the game of golf. To do this I am going to have to model and represent a complete 3d world. There are several distinct parts of this project that I have identified. These are as follows:

- **Map Contour** - The logical modelling and representation of a three-dimensional contoured terrain. (The golf course)
- **Map Texture** - The modelling of different textures of the terrain and how the textures affect the ball. Textures include short grass / green, medium grass / fairway, long grass / rough, sand and water.
- **Real World Objects** – The modelling and representation of objects (such as trees and bushes). I will have to model the effect they have on the ball when the ball hits them as well as finding an efficient and realistic way to display them.
- **Ball Travel** - The modelling of the ball. This covers the flight of the ball, the effect of impact on the ball and the rolling of the ball. Factors to be taken into account include gravity, air, wind, effects of spin etc.
- **Clubs** - The modelling of different types of golf club, the different effects they can have on the ball in terms of spin, power and direction.
- **3D Engine** - The three-dimensional display engine. Various aspects need to be implemented, such as translation of 3d world points into 2d screen points, drawing of filled 3d polygons and rendering scenes (displaying closer objects over distant object etc.).
- **Game Engine** - The implementation of game rules. Enabling multiple players to compete against each other. Time permitting possibly include computer opponents of variable skill levels and maybe even network or internet play.

There are also some separate small projects that I plan to design and build as follows:

- **Ball Flight Tester** - Compromises of 3 simple 2d representations - top, front and side used to help develop and test the algorithms that I will develop to model the flight of the ball.

- **Map Representation** - This will be built initially as a generic 3D engine but will be used to develop the data structures and algorithms for the development of the contoured golf course.

For more information about my initial plans for this project please view my project proposal in Appendix A.

2. Project Definition

The Project definition is an overview of the main aims and objectives of this project.

The overall aim is a description of what the final product will be. It does not cover anything to do with how I am going to go about reaching this goal.

The objectives cover the sequence of goals that I aim to complete during this project. This includes the processes I plan to go through, not just objectives of the product.

2.1 Overall Aim

The aim of this project is to create a playable simulation of the game of golf. It will simulate the laws of physics as accurately as possible whilst being an enjoyable and playable game.

2.2 Objectives

- Research the possible ways to model and represent the various separate aspects of the system (map contour, map texture, objects, the ball etc).
- Produce a requirements document to encompass every aspect that the system must implement.
- Assess the Risks involved in this project and produce a risk document.
- Design the architecture of the software, the class definitions, the sequences of events, the data structures etc..
- Implement the Tester Application. Use it to create and test the algorithms used in the manipulation of real world objects.
- Implement and test the game itself.
- Create technical documentation to explain the code for future maintenance.
- Create a user manual.

3. Project Plan

3.1 Work Breakdown

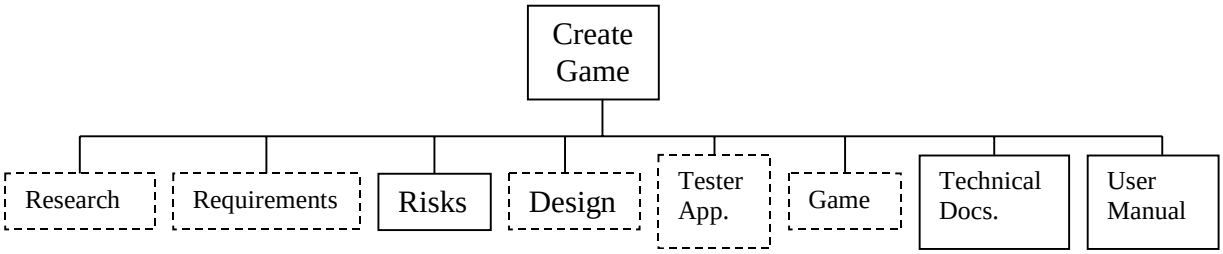


Figure 3.1.1. The initial work breakdown structure.

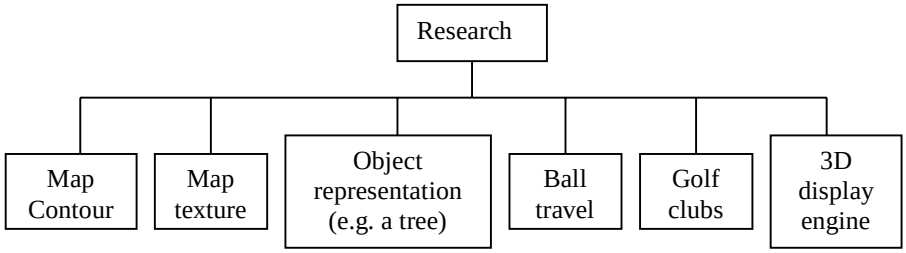


Figure 3.1.2. Further work breakdown of Research.

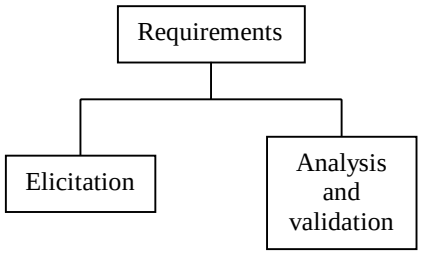


Figure 3.1.3. Further work breakdown of Requirements.

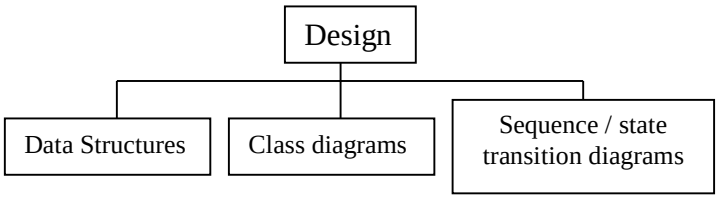


Figure 3.1.4. Further breakdown of Design

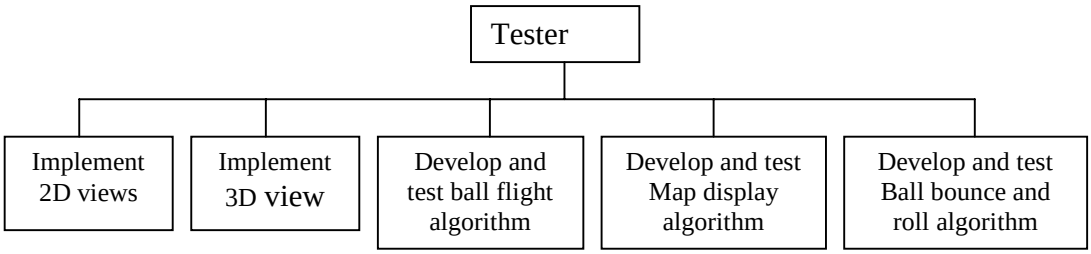

Tester
Implement 2D views
Implement 3D view
Develop and test ball flight algorithm
Develop and test Map display algorithm
Develop and test Ball bounce and roll algorithm

Figure 3.1.5. Further work breakdown of the Tester App.

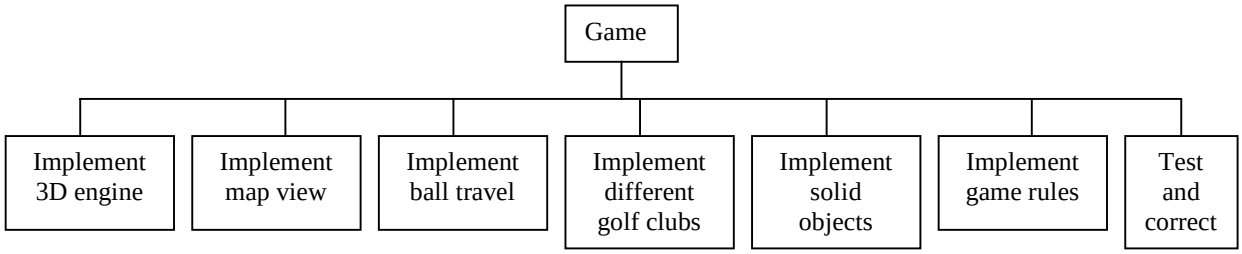


Figure 3.1.6. Further work breakdown of the Game.

Figures 3.1.1 to 3.1.6 show the breakdown of the work that needs to be done in order to complete this project. Figure 3.1.1 is the head of this structure with the rest being extensions of it. The reason for these diagrams is to enable me to plan the stages that the project will go through.

In the majority of information systems development cycles, a major part of the project is the analysis and the modelling of the data and practices of the system they will be implemented to support or replace. It is evident from the above diagrams that I have skipped past that phase. This is because I am not replacing or supporting any established system, I am simulating a real world situation. This means that there is no modelling to do because all the information required will be gathered from the research.

3.2 Time Estimation

The next step in creating a good project plan is to estimate how long each of the tasks I have identified will take. Figure 3.2.1 shows the time I have allotted to each low-level task.

Activity	Estimated Duration
Research:	
Map Contour	1 weeks
Map Texture	1 weeks
Object Representation	1 week
Ball Travel	1 weeks
Golf Clubs	0.5 weeks
3D display engines	1 weeks
Requirements:	
Elicitation	2 weeks
Analysis and validation	1 week
Risk	1 week
Design:	
Data structures	3 weeks
Class Diagrams	1 week
Sequence / state diagrams	2 weeks
Tester App:	
2D views	0.5 weeks
3D view	1 week
Ball flight	1 week
Map display	1 week
Ball bounce and roll	1 week
Game:	
3D Engine	2 weeks
Map view	2 weeks
Ball travel	1 week
Golf clubs	0.5 weeks
Objects	1 weeks
Game rules	2 weeks
Testing and correction	2 weeks
Technical documentation	1 weeks
User manual	1 week
Total	32.5 weeks

Figure 3.2.1. Time estimations for all planned project stages

The purpose of these time estimations is to help me to organise the sequence that I will follow whilst I am working through this project. It is also useful in identifying areas that may be too complex. This may be due to having over complicated your project, in which case, your

project and hence the plan would have to be amended. I have made one major change to my plan for this project since the proposal, that is that I will not be using the rational unified process during this project. The reason for this is that I feel it will overcomplicate this project and the control that it offers is overshadowed by the amount of extra time it will take.

3.3 Milestones

Milestones represent significant steps towards the completion of a project. They also act as a short-term target to aim for. The state of a project can be very quickly found be assessing the progression of completed milestones. The milestones that I have identified for this project are as follows:

(M1) Research completed.

(M2) Ready for design (Research, Requirements and Risk completed).

(M3) Application design completed.

(M4) Main algorithms developed (Tester app. has been used to create all useful algorithms)

(M5) Game is playable.

(M6) Project completed.

These milestones show a good overview of the main stages that the project will go through. The ordering of milestones here does give a good representation of which order they will be completed in (the only possible exception would be that M3 and M4 could be transposed). This ordering is not sufficient enough to complete the project plan though, so a more in depth activity sequence is required.

3.4 Activity Sequence

The sequencing of activities is a very important part of the project plan. It gives a clear network of paths that should be followed. It enables me to ensure that all work needed before a new activity can begin has already been completed. It also helps me to know what tasks can be done concurrently. This is very important, as having more than one thing that you can work on is more efficient. This is because, if I got stuck on something, I always have something else I can work on and come back to the problem fresh at a later time.

Figure 3.4.1 shows the activity sequence for this project. There are a few things that are not included in this sequence. These things are the proposal, the project plan itself and the report write up. The reason for not including the proposal and the plan are that they have already been done. And I do not feel that they are actually a part of the project, they seem more like stages you go through before you actually start the project itself. The reason for not including the report writing is that it is not a stage, it is something that happens at every stage and so it is basically inferred.

The actual hand in date for this report is May 4th. The activity sequence suggests that the project will be finished by March 22nd. Unfortunately, I do not believe that the project will finish on time and so the extra gap at the end will be a very useful buffer. Even if the project does finish within scheduled time, the gap at the end will serve to either polish up parts that I am unhappy with or maybe even give me a bit more spare time to revise for my exams!

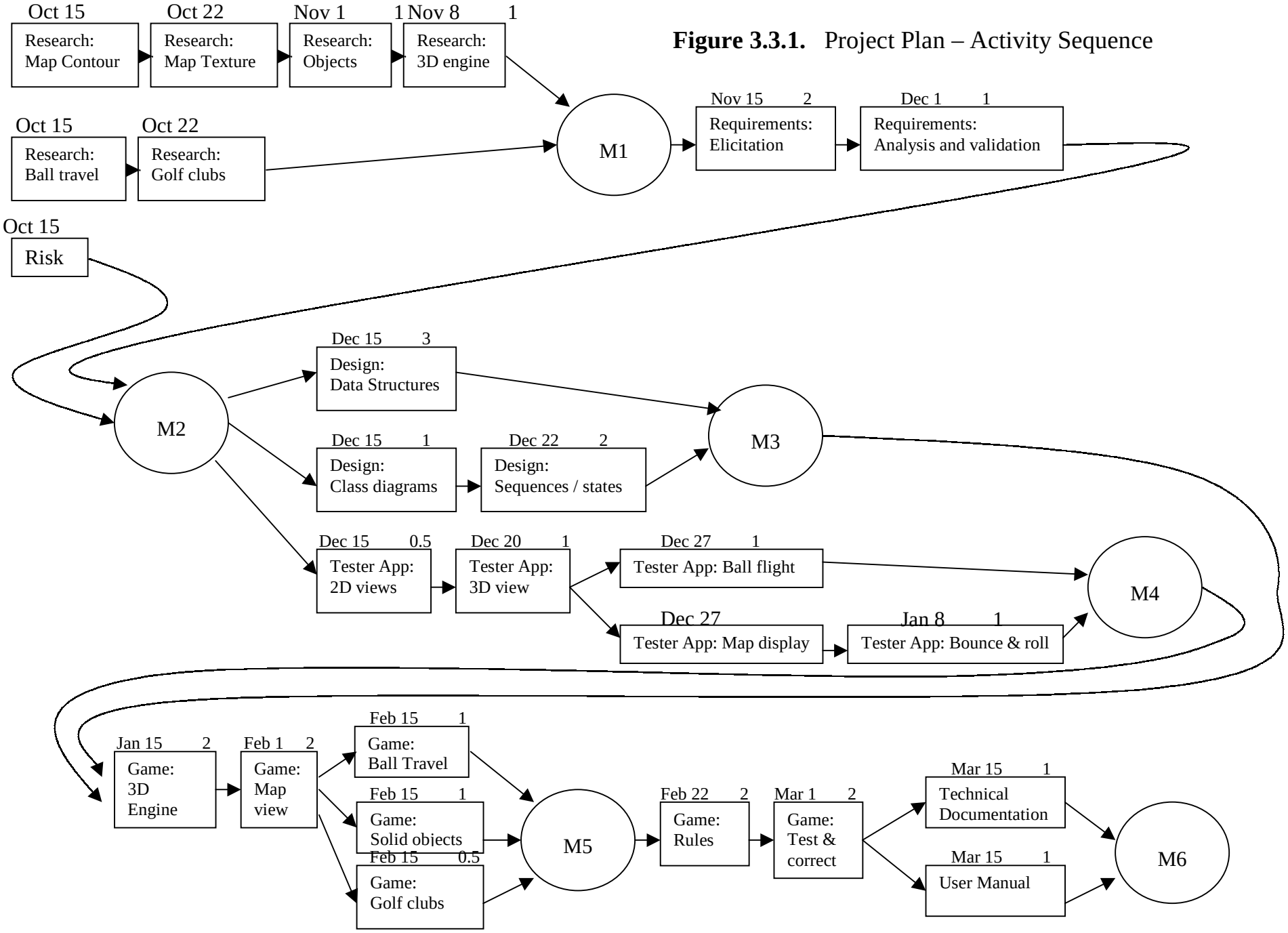


Figure 3.3.1. Project Plan – Activity Sequence

4. Research: Map Contour

4.1 Mathematical representation of curves.

Any curve can be represented using mathematical formulae. This has some very distinct advantages, the main being the accuracy. A curve represented by a formula is perfectly smooth and accurate and allows any point on a surface to be calculated to any degree of accuracy required. Curves represented in this way will never look blocky. The most common example of mathematical curves is a True Type Font. Figure 4.1.1 shows the difference between a True Type Font and a fixed pixel font.

Figure 4.1.1. Showing the difference between 2 enlarged fonts.
(A) is “System”, a fixed pixel Font.
(B) is “Arial”, a True Type Font.

As you can see, the True Type Font still looks very smooth even when enlarged. The fixed pixel type font represents straight lines just as well as the True Type Font (The “L” and “F” look perfect) but the curves (the “G” and the “O”) are excessively blocky and inaccurate.

In an ideal world, the mathematical representation of the terrain would be perfect. However, the complexity involved in creating the terrain this way is far too much of an overhead. Not only is it incredibly difficult to create a terrain this way, but it is far too complex to handle the display from a 3D mathematical model to a 2D screen. The overhead in time, not just in the creation but also in the translation to screen, would make a “Real Time” 3D engine incredibly slow.

4.2 Freeform polygonal mesh.

Polygonal meshes are the most commonly used method of representing three-dimensional objects. They are composed of multiple flat polygons, arranged in such a way as to approximate the shape they are representing. Polygons are perfect for representing flat objects such as doors and walls. Curved objects, such as spheres, can only be roughly approximated. The accuracy of a curve represented by polygons depends on the number and size of the polygons. Figure 4.2.1 shows how the frequency of polygons can affect the accuracy of the shape.

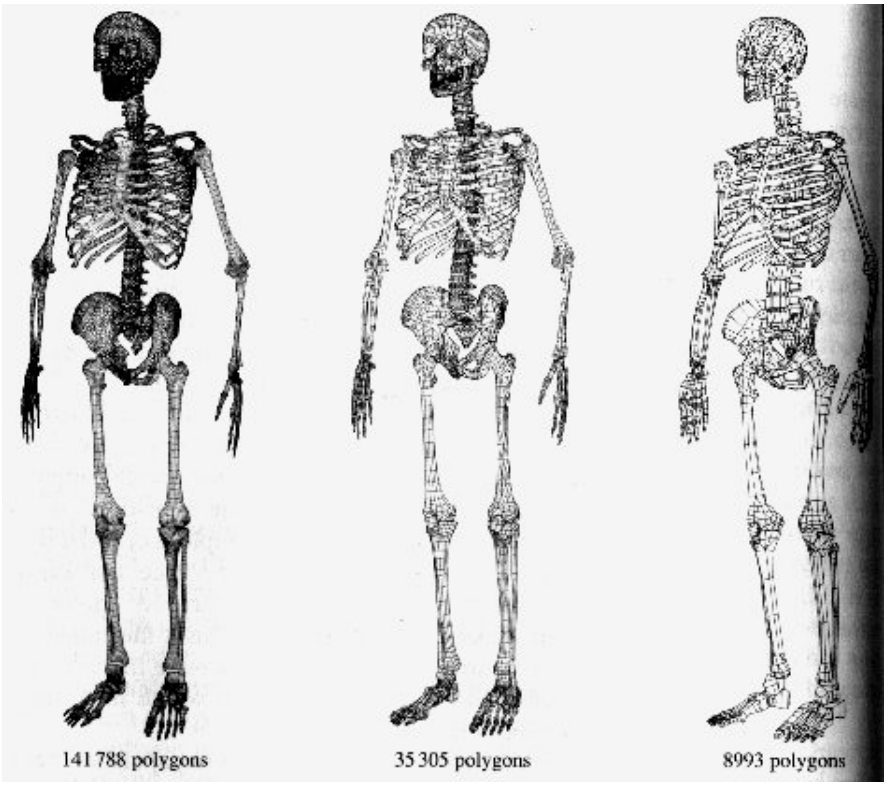


Figure 4.2.1. Showing how the frequency of polygons affects the accuracy of the representation. *Taken from "3D Computer Graphics" by Alan Watt*

It seems fairly obvious that the more polygons, the more accurate the shape. This is absolutely true, however, the more polygons, the more complex the object is. This makes the modelling, manoeuvring and representing of this object much more time consuming with more complex objects. The decision on how complex to make your object is not a straight forward one and may involve striking a happy medium. There are also ways to make less complex

objects look more realistic. Texture mapping allows simplified and rough polygonal objects to seem like they are far more detailed. Figure 4.2.2 demonstrates this. See **5.2 Texture Mapping** for more detailed information.

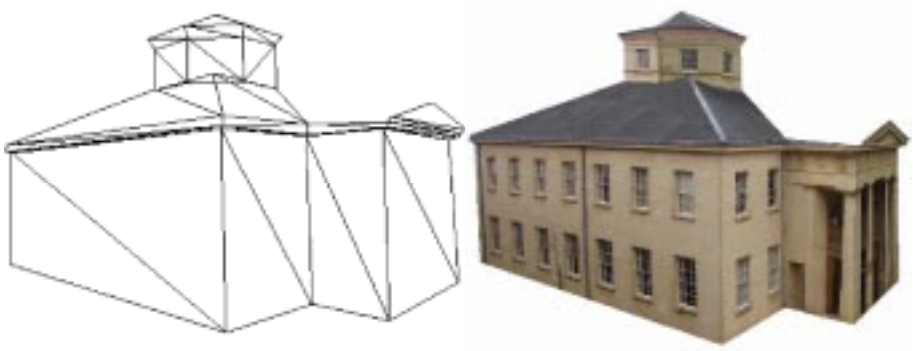


Figure 4.2.2. Texture mapping can make a rough representation look very realistic.
(Pictures from University of Cambridge Department of Engineering http://svr-www.eng.cam.ac.uk/research/vision/photobuilder/recon.html)

The main advantage to the freeform polygonal mesh is that it can represent almost anything. However, when it comes to collision detection (detecting when 2 objects have come into contact), there is no simple method. There are several methods that can be used. These methods include Bounding Spheres (See Fig. 4.2.3) and Intersections (See Fig. 4.2.4).

Bounding Spheres is a method in which each polygon is surrounded by a theoretical sphere that contains every point of that polygon. Collision or possible collision is detected when the distance between 2 objects (possibly a single polygon) is less that the sum of the radii of the objects.

Intersection based collision detection works by checking if any edges of one polygon pass through the surface of another. This is a complex process and so is most often made into a 2 stage process. The first check is to see if the any edge of the first polygon passes through the plane of the second. This is a much faster process and acts as a check to see if the collision was possible, if it was then continue with the second check – checking if the intersection is within the bounds of the second polygon.

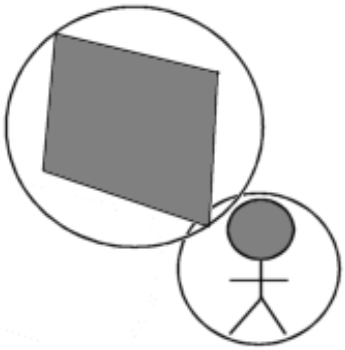


Figure 4.2.3. Collision Detection using Bounding Spheres.

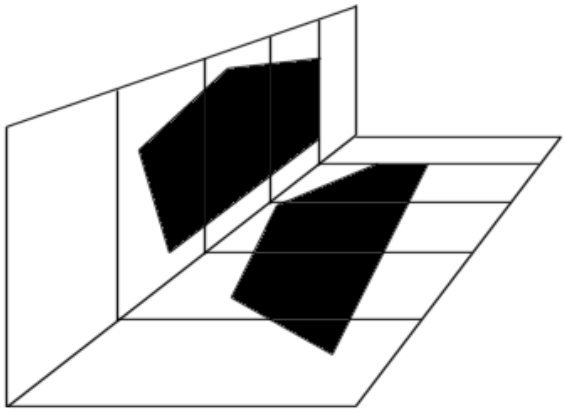


Figure 4.2.4. Collision Detection using Intersection. Here, there has been no collision.

Both of these methods are considerably time consuming and surely there must be a more simple method with a shape as simple as a golf course. This brings me to my own idea – a polygonal mesh with a set structure.

4.3 Structured polygonal mesh.

This is an idea of my own. My idea here is that the terrain is built out of polygons that follow a strict set of rules. These being that the vertices adhere to a strict grid with only the height being variable, fig 4.3.1 shows what I mean. The most obvious disadvantage of this type of representation is that it is very inflexible. It can be made more accurate by having smaller gaps between the grid but it will still lose accuracy when used to represent steep surfaces (see fig 4.3.2).

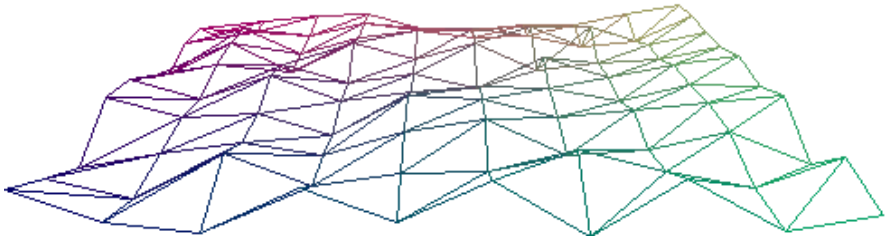


Figure 4.3.1. A depiction of "Structured Polygonal Mesh"

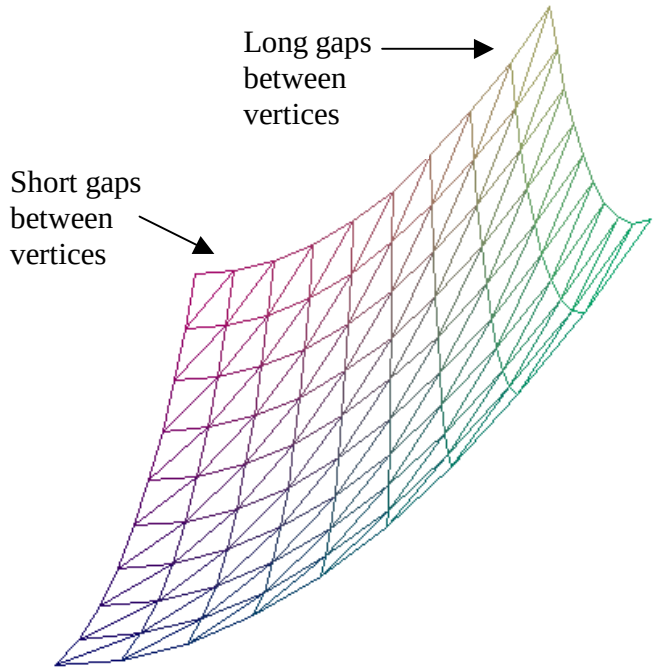


Figure 4.3.2. Shows how the accuracy is lessened on slopes.

There is, however, a huge advantage to this way of representing the map and that is the collision detection. In an all purpose 3D engine it may not have much of a benefit, but in this program, the only thing that is going to collide with the map is a single golf ball which is probably going to be modelled as a single point. What this means is that with the map structured in this way, you can work out exactly what polygon is directly beneath or above the ball simply by knowing the coordinates of the ball on the horizontal plain. This means that to detect when the ball has hit the ground, you only have to check a single polygon. This will make the game much faster and a lot more feasible to write in Java.

I had 2 different ideas for what the structure could be. Figure 4.3.3 demonstrates the 2 different structures. Structure (a) seems to be a lot more adaptable to different type of curves than (b) but would be a lot more difficult to store and manipulate the map data. Structure (b) is a far simpler structure but could cause more unnatural looking results than (a).

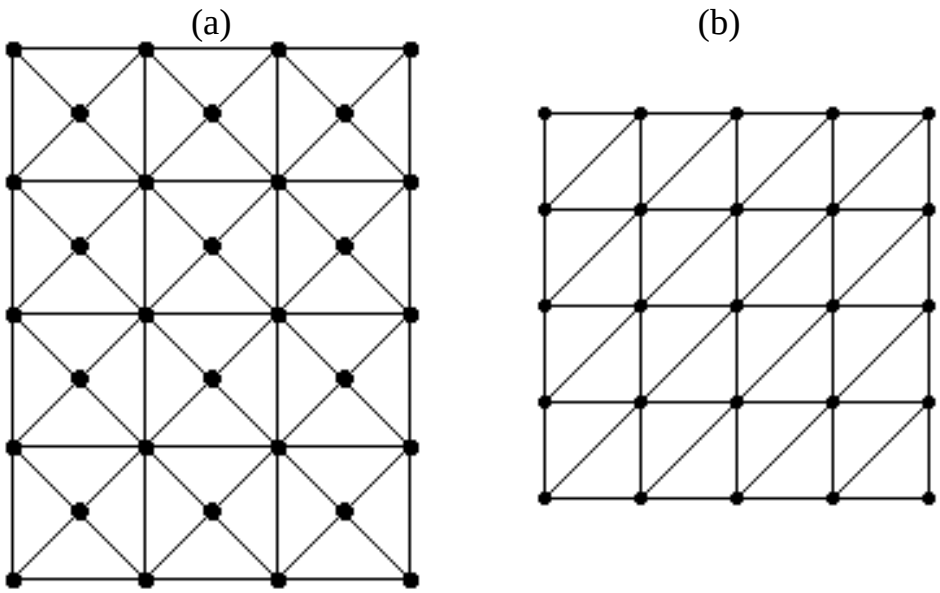


Figure 4.3.3. Two possible mesh structures. The large dots indicate the stored vertices.

5. Research: Map Texture

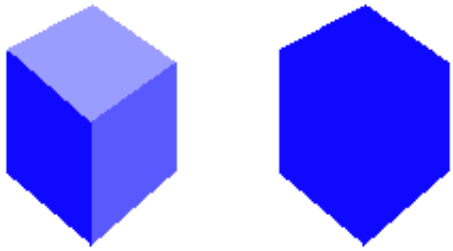
This part of my research actually covers 2 topics:

1 The display of the map texture.

2 The interaction between the ball and the different surfaces.

5.1 Light Shading

Light shading is needed to show the shape and contour of an object that has a solid colour. Fig 5.1.1 shows 2 cubes, a) is light shaded while b) is not. As you can see, light shading varies the colour of an object depending on the orientation of the surface. This results in the shape of the object being visible, as object b) shows, without any light shading, the shape of the object is completely lost.



a) Light Shaded b) Solid Colour

Figure 5.1.1. Showing the need for light shading.

There are various light shading methods that can be implemented. The most simple is as shown above where each polygon is given a solid colour (this is sometimes referred to by many names, most usually though it is Face or Facet shading). This is a very fast method, and is very effective for shapes with flat surfaces, such as a cube. However, for curved surfaces, where the polygons are used to approximate the curve, this method produces unrealistic results with very bold lines between polygons. To account for this, there are alternative, more complex, shading algorithms available. The most popular of which are Gouraud shading and Phong shading. The basic idea behind both of these is to calculate the colour at the vertices of each polygon and then vary the colour across the polygon so that you get smooth transitions between vertices. What this means is that adjoining polygons, which share vertices between them will be shaded smoothly from one into the other, thus eradicating the obvious differences between adjoining polygons and producing a smooth, curved effect. Fig 5.1.2 shows a comparison between Gouraud and Phong shading.

Facet / Face Gouraud Phong

Figure 5.1.2. A comparison between Facet / Face, Gouraud and Phong shading. (Picture is from the New York Institute of Technology Computer Graphics Lab)

5.2 Texture Mapping

Texture mapping is where an image is overlaid onto a 3D object, often simply a single polygon. As I mentioned previously (4.2 Freeform Polygonal Mesh), it is a way to make very roughly modelled shapes look far more detailed (See fig 4.2.2).

The process behind texture mapping involves loading a 2D bit mapped image into memory, associating the x,y coordinates of the polygon to the x,y coordinates (often referred to as u,v coordinates but they mean the same thing) of the relevant part of the bitmapped image. Each pixel of the polygon is then mapped onto a pixel in the stored image, and the colour of the stored image is then used as the colour of the polygon. This description is merely an overview of an incredibly complicated process and is by no means complete.

5.3 Bump Mapping

> *"in 2-D texture-mapping when a single digital image of a rough surface is mapped onto a 3-D object, the appearance of roughness is lost or distorted."* **Dana et. Al. [1999]**

This is because a non-smooth surface looks very different when viewed from different angles. Even slight bumps in the surface could cover part of the surface when viewed from an acute

angle. One solution to this problem is Bump Mapping. This entails the holding of additional information for each polygon. This information is basically a height mapping that describes the very small-scale differences in height across the surface of the polygon.

Becker and Max [1993] state:

> *"Bump-mapping is an inexpensive way to achieve a good approximation to a macroscopic surface roughness."*

Notice the word "Inexpensive", what they mean is that it is a rather simple process when compared to others available (See 5.4 Bi-directional Reflectance). "Inexpensive" meant in the way of speed and memory consumption as opposed to price. However, it is still a very time consuming process when compared to simple light shading and texture mapping.

5.4 Bi-directional Reflectance and Texture

Dana et. al. [1999] state:

> *"Bump-mapping preserves some of the appearance of roughness, but knowledge of the surface shape is required and shadows cast from the local surface relief are not rendered."*

Dana's paper on Reflectance and Texture introduces and discusses a new texture representation called the BTF (Bi-directional Texture Function). The roots of this are based firmly in the BRDF (Bi-directional Reflectance Distribution Function) Introduced by Nicodemus [1970]. For the BTF, they have gone to the extent of building a database of images taken from 60 surfaces observed from over 200 combinations of viewing and illumination angles. The reason for this is that a single image for texture mapping is insufficient. In the real world, viewing a texture from an angle causes much of the texture to be obscured by the peaks in the texture, also, shadows are cast from these peaks and the effect of these shadows depends on the lighting conditions. Single image texture mapped objects just show a distorted version of how the texture looks head on.

Any engine built to use the BTF or BRDF is guaranteed to be painfully slow. Far too slow in fact to even consider using them for my golf simulation. However, their use is in applications that require incredibly accurate images to be produced regardless of time taken.

5.5 Surface Interaction

It is not just the displaying of the various surfaces that I must be concerned about. In fact, more importantly in this project is the realistic modelling of the interactions with the surfaces. I could talk about general properties of surfaces, such as hard surfaces causing the ball to bounce higher than softer surfaces. However, surfaces such as long grass are more complex than that, the ground underneath may be hard but the grass itself may trap the ball. I have identified the following distinct types of surface and have described the properties of each.

"Green"

This is very short grass. The surface has very few major deviations and so is generally very smooth. This tends to be a very hard surface, and causes the ball to bounce relatively high. It also has very little resistance to rolling. When the ball is struck from this surface, it will travel away very accurately because there is little to get in the way.

"Fairway"

This is medium length grass. This surface may have some considerable deviations but is on the whole relatively smooth. The ball will bounce fairly high off this surface as it is fairly hard and the grass is not too long. The ball will also roll well across this surface. Striking the ball from the fairway will result in a very accurate shot.

"Rough"

This is long grass or other very rough / course material. This surface will absorb a lot of the energy of the ball and will cause either very low bounces or no bouncing whatsoever. A ball will not roll very far through the rough because of the length and density of the grass. The grass enveloping the ball will also cause any strokes taken from the rough to be extremely inaccurate.

"Sand"

Literally meaning sand, as found in bunkers (sand traps). When the ball lands in sand, almost all of the energy should be absorbed so the ball will either not bounce or will bounce very low. Sand will not offer as much resistance to rolling as rough will, but the ball will definitely not roll very far across sand. Strokes taken from sand will be relatively accurate but will have to be very lofted to help the ball escape from the sand otherwise it won't really go anywhere.

"Water"

It is very rare that a ball struck into water will escape. For this reason, it is not necessary to model the properties of water, I will simply assume that the shot is over as soon as the ball contacts water.

5.6 Decisions

For the purpose of this project, I believe that simple light shading of solid colour will be sufficient. This will result in the golf course not looking quite as realistic as it could have but the graphics engine will run a lot quicker. It will also be a lot quicker to develop, and with time being so limited, this is probably the most sensible choice. The easiest way to achieve this is to have a single light source (the sun) and vary the colour of the polygons that make up the golf course depending on the angle between them and the light. This loss in realism is acceptable because the main focus is on the realism of the implemented physics rather than the display engine.

6. Research: Ball Travel

The modelling of the ball travel is a major part of this project. This includes calculating the flight of the ball, the bouncing of the ball and the rolling of the ball. Each of the major aspects that affect these things are researched below.

6.1 Equations of motion

There are a few simple equations that are used to calculate linear motion. These are as follows:

$v = u + at$
$u = v - at$
$s = ut + \frac{1}{2} a t^2$
$s = vt - \frac{1}{2} a t^2$
$v^2 - u^2 = 2as$
$t(u + v) = 2s$

Where u = Initial velocity
v = final velocity
a = acceleration
t = time
s = distance

These are extremely useful and can be used to calculate the position and velocity of a particle after any specified length of time. They are intended for use in a single direction, however, the movement of a projectile can be modelled in all three axis independently (eg, the x,y and z coords).

6.2 Gravity

The true equation for gravity takes into account the mass of 2 bodies and the distance between the centres of each. This would be quite complex, as I would need information about the size and mass of the earth. Fortunately, I will not need to use that equation. The reason for this is that due to the immense size of the earth, the acceleration due to gravity (g) varies by less than one percent within 30km of the Earths surface (De Mestre 1991). For this reason, we can disregard the mass of the Earth and the distances and simply use the following equation:

$F = mg$

Where F = Force acting on the ball in Newtons
m = Mass of ball in Kg
g = Acceleration due to gravity = $9.8ms^{-2}$

If gravity was the only force I would take into consideration, I could just ignore the force and just use the fact that the ball will be accelerating at $9.8ms^{-2}$ towards the ground. However, as I will be considering other factors, I will need to keep track of the magnitude and direction of all forces so that I may discover the resultant force from all affecting factors.

6.3 Air and Wind

When a projectile travels through any fluid (gas or liquid), resistance will cause the projectile to decelerate and eventually stop. This "Drag" force is directed in the opposite direction to the direction of travel of the projectile. In practice, the magnitude of this force is not directly proportional to the speed of the projectile, however, De Mestre (1991) states that at low speeds through low density fluids (such as air), the drag can be assumed to be directly proportional to the speed without significant loss of accuracy.

This means that the calculation for the drag force is much simpler than it could have been. The following formula can be used:

$F = mkv$

Where F = Drag force in Newtons
m = mass of projectile in Kg
v = velocity of the projectile in ms^{-1}
k = resistance coefficient per unit mass (unti s^{-1})

This equation is over simplified for general use because it doesn't take into account the shape and texture of the projectile. It is fine for use in this project because the ball is assumed to be perfectly symmetrical and I am not going to be modelling this for any other objects.

Wind is very easy to model, all that the wind does is to change the speed of the ball relative to the air. What this means is that to account for the effect of the wind, all that has to be done is to discover the speed relative to the air rather than to the ground and use this speed when

calculating the effect of air resistance and spin. Fig 6.3.1 shows 2 examples of the speed of a ball relative to the air in wind.

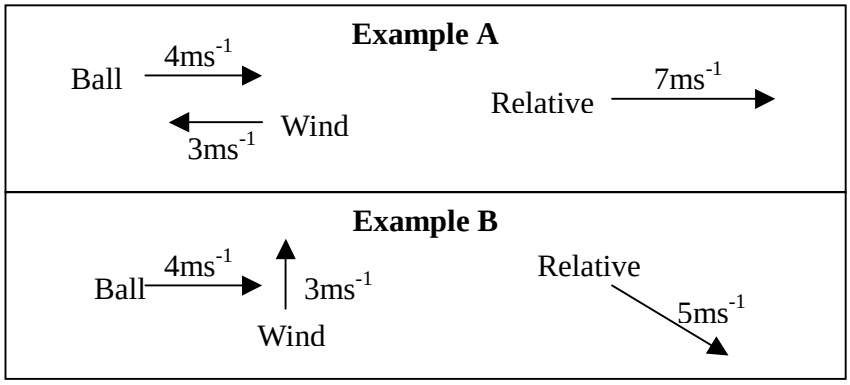


Figure 6.3.1. Shows the speed of a ball relative to the air, accounting for wind.

6.4 Density, Temperature, Pressure & Humidity

Density, temperature, pressure and humidity are all factors that affect the resistance coefficient of the air. However, for the purposes of this simulation, we can simply assume that these factors will be constant. Doing this simply enables a single constant value for the resistance coefficient to be set and we can then forget about density, temperature, pressure and humidity.

6.5 Effects of spin in flight

One of the major factors that affects the trajectory of a ball / projectile is spin. If you have ever watched any ball sport you will have noticed this. Footballers use spin to curve the ball around players, and, tennis players use topspin to force the ball to dip so that they can hit the ball hard without it soaring out of the court. In the game of golf, spin can be imparted onto the ball by slicing across the ball as it is being struck. This spin greatly affects the way the travels both in the air and on the ground, so, I have decided to investigate the ways in which spin affects the ball in flight.

Fig 6.5.1 shows a typical situation with a ball travelling through the air with an anti-clockwise spin. The main thing to notice here is that on the left, the spin of the ball is making the

surface of the ball move in the same direction as the air, whereas, on the right, the surface of the ball is moving in the opposite direction to the air.

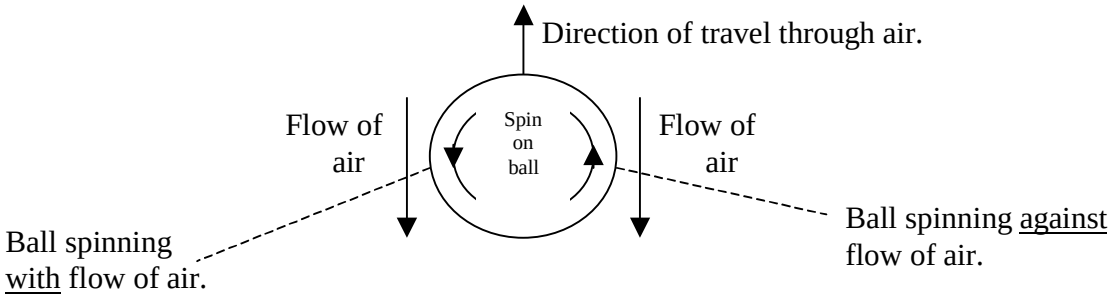

Direction of travel through air.
Flow of air
Spin on ball
Flow of air
Ball spinning with flow of air.
Ball spinning against flow of air.

Figure 6.5.1. Typical situation of ball flying through air with anti-clockwise spin.

The movement of the surface of the ball against the air causes the speed of the air to change. On the right hand side, the surface of the ball moving against the air causes the air to rapidly decelerate, whereas, on the left the air will be decelerated less or possibly even accelerated. It is a widely known fact the slower the air is moving, the greater the pressure it exerts, this means that the pressure exerted by the air on the right is greater than that exerted on the left. Fig 6.5.2 demonstrates this fact. This difference in pressure causes a resultant force in the direction from the high pressure to the low pressure. This means that the resultant force from spin is perpendicular to the direction of travel through the air, as demonstrated in fig 65.3.

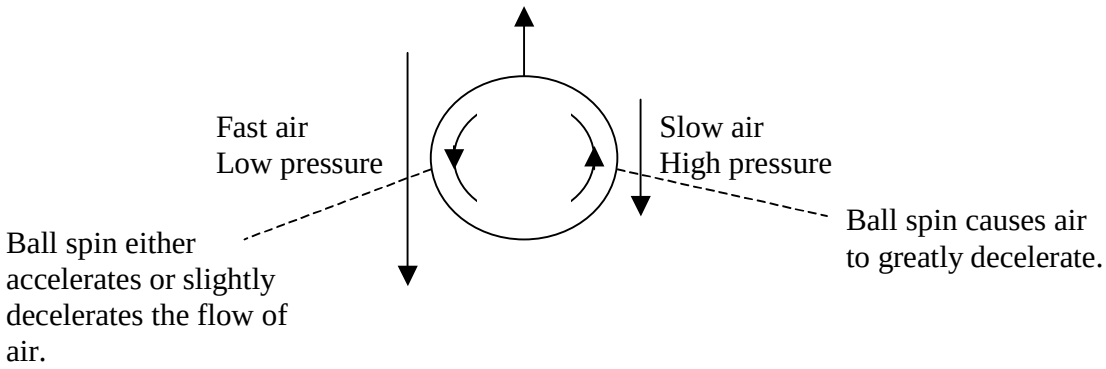

Fast air
Low pressure
Slow air
High pressure
Ball spin causes air to greatly decelerate.
Ball spin either accelerates or slightly decelerates the flow of air.

Figure 6.5.2. Showing the differing air speeds and hence pressure of a spinning ball.

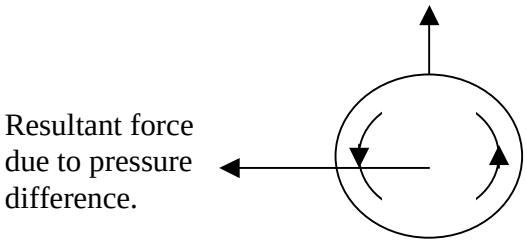


Figure 6.5.3. Showing the resultant force due to spin on a ball in flight.

A sphere will spin on any axis that passes through its centre. This could be modelled by storing the horizontal and vertical angles of the axis along with its angular velocity. However, this data is very difficult to work with. Instead, as I will be modelling the flight of the ball horizontally separate from vertically, the spin of the ball can be successfully modelled as 2 different values. These are its horizontal (left/right) spin and it's vertical (top/back) spin.

6.6 Contact with ground

To effectively simulate the bouncing of the golf ball, I have had to investigate the physics behind bouncing. Fig 6.6.1 shows what happens when a ball strikes the surface. If the ball and surface were perfectly flat, hard and frictionless, the ball would bounce off the surface with the angle of reflection equal to the angle of incidence. This is far from realistic though, when the ball strikes the surface, the surface absorbs a lot of the ball's energy and also alters the ball's trajectory. As Fig 6.6.1 shows, the realistic new trajectory is more towards the normal to the surface than a perfectly reflected bounce.

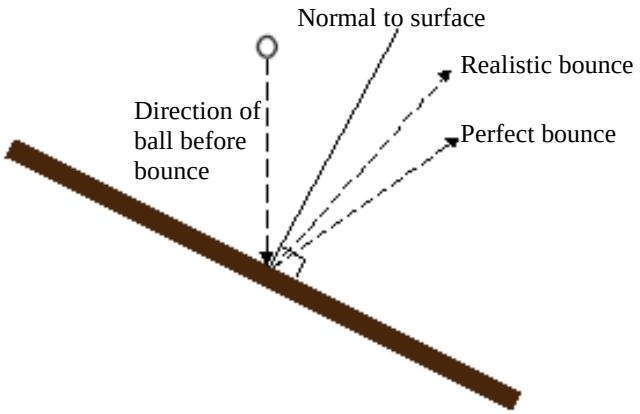


Figure 6.6.1. Showing the direction of a ball immediately after bouncing.

The amount that the trajectory is altered depends on the texture of the ground. Fig 6.6.2 shows the general effect that different surfaces have. There are a lot more factors that come into consideration such as the composition of the surface – is it sticky, is it powdery etc. However I feel that this is far too complex a subject to model in too great a depth so I am going to stick with the hard-soft factor.

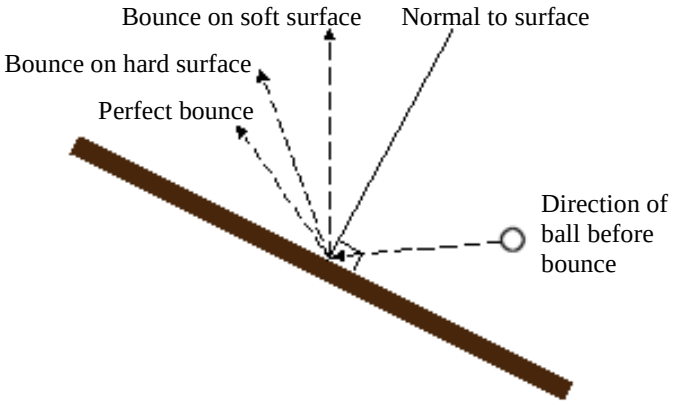


Figure 6.6.2. Showing the effect that the surface texture has on the direction after bouncing.

6.7 Rolling

The final area of investigation as far as the travelling of the ball is concerned is the investigation into the factors that affect rolling. These factors include slope, spin and friction. Fig 6.7.1 shows the effect of the ball being on a sloped surface.

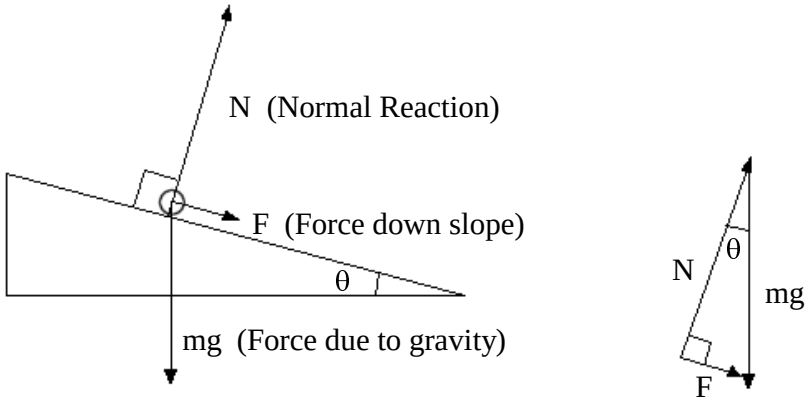


Figure 6.7.1. Showing the force on a ball down a slope due to gravity.

As you can see from the diagram shown in Fig 6.7.1, gravity is responsible for the effect of the ball accelerating down the slope. As we will only ever be dealing with the movement of one object (the ball), we can assume m (Mass) to be 1, therefore mg = g. We can now deduce the formula to calculate the resultant force directed down the slope as follows:

$$\sin\theta = \frac{F}{g}$$

$$\therefore\ F = g\sin(\theta)$$

Another of the main forces to consider when modelling the rolling of the ball is friction. Friction is a force that arises due to resistive forces between the moving object and the surface it is moving over. It is directed in the opposite direction to the motion of the object. The magnitude of this force is relative to the normal reaction on the object from the surface. If you look back to Fig 6.7.1, you will see that on a sloped surface, the normal is calculated as:

$N = g \cos(\theta)$ (again assuming that Mass = 1)

Whereas, on a flat surface, N will simply be equal to g. This can be proven since $\cos(0) = 1$.

The equation to calculate the magnitude of this force is:

$F = \mu N$

where μ = The coefficient of friction between the 2 specified surfaces.

The final factor affecting the rolling of the ball is spin. The types of spin and the relevant effect is shown below:

Type of spin	Force
Top	Forward
Back	Backward
Right	Very Slight Right
Left	Very Slight Left
Top & Right	Forward & Right
Top & Left	Forward & Left
Back & Right	Backward & Right
Back & Left	Backward & Left

As you can see, sidespin alone causes very little force until it is combined with topspin or backspin. One very important fact however is the fact that spin is very quickly lost when sliding / rolling across a surface with relatively high resistance such as a golf course. The spin on the ball very quickly gets turned into topspin, causing the ball to roll rather than slide.

7. Research: Golf Clubs

The key to getting a shot right is in choosing the right club. To implement a realistic way to simulate the effects of different types of golf club, I need to investigate what types of club there are and what the differences between them are. I will also take a look at why you might choose different clubs in different circumstances so that the computer could make a good guess at what club to automatically suggest.

7.1 Types of Club

There are 4 main classes of club. These are Wood, Iron, Wedges and Putters.

7.1.1 Wood

The wood clubs have a very flat face. They are used usually to tee off because they make the ball travel the furthest. They are available in a number of flavours, usually from 1-Wood up to 5-Wood. The lower the number, the flatter the face of the club (see fig. 7.1.1) and hence the further it can drive the ball. The 1-Wood is more often referred to as "The Driver".
Woods in general produce between 8 and 12 degrees of loft (0 degrees of loft would be perpendicular to the ground).

Vartan Kupelian from golfonline.com talks about the rapidly expanding area of "Trouble Woods", these are Woods numbered 6,7,9 and 11. These clubs are not commonly used but are preferred by some to the longer Irons because they feel they are easier to hit.

NB. Just because they are called wood doesn't mean they are made from wood!

7.1.2 Iron

The iron clubs have a more angled face than the wood clubs. This causes them to give the ball a much higher flight than a wood club would. The usual types of irons are 1-Iron down to 9-Iron with the higher the number, the more angled the face.
NB. Again, they will probably not be made from iron!
<u>Wedges:</u>

Wedges are again a progression in the amount of angle on the face of the club. They are seriously angled and hit a lot higher that they do far. They are usually used in areas where the ball is in a particularly bad lie, that is, that it is surrounded by grass, sand etc. The usual flavours are simply pitching wedge and sand wedge.
NB. Wow! They do actually look like a wedge!

7.1.3 Putters

These are relatively simple clubs. They usually have a completely flat face so that they can hit the ball without it being chipped into the air. There are not many types of putter available, they all do the same type of thing. The only major difference in putters is that some have a much longer handle to enable a different grip, this has no effect on the ball, other than giving the player a more comfortable or accurate swing.
NB. They are actually used for putting!

Wood / Driver Iron / Wedge Putter

Figure 7.1.1. Rough representation of the different profiles of Golf Clubs.

7.2 Effect on Ball

Table 7.2.1 shows a guide as to how far different clubs can hit the ball. Obviously it depends greatly on the ability of the player and on a whole plethora of other factors, it is, however, only intended as a rough guide.

Type of club	Average distance of full swing
Driver (1-Wood)	200-220 mtrs
3-Wood	180-200 mtrs
5-Wood	170-190 mtrs
3-Iron	175-185 mtrs
4-Iron	155-165 mtrs
5-Iron	145-155 mtrs
6-Iron	135-145 mtrs
7-Iron	125-135 mtrs
8-Iron	115-125 mtrs
9-Iron	105-115 mtrs
Pitching Wedge	85-100 mtrs
Sand Wedge	65-85 mtrs

Table 7.2.1. Average ball travel distances caused by a range of Golf Clubs

7.3 When to use which club

The main issue to consider when choosing a club is the distance from the ball to the hole. The information in Table 7.2.1 gives a rough guide as to which club corresponds to which distance. However, there are a few more factors to consider than that. The lie of the ball is also very important. For example, if the ball is in heavy rough, surrounded by long grass, you would not use a driver to hit it. What you should do is to select a club that will loft the ball high in the air, to escape the grass as soon as possible.

The surrounding landscape must also be considered when selecting a club. The distance to the hole may be fairly long but you may have a steep face directly in the way that you will have to avoid. In this case, you will probably select a much higher valued iron or a putter and take the risk of under hitting the shot.

You must remember however that you are only allowed to take 14 clubs with you on any game. This means that you will not be able to take the full range of clubs. The clubs that you select should be scattered throughout the range of available clubs to ensure that you have a club for almost any situation. You should always however, carry some kind of wood for teeing off and a putter for putting.

8. Research: 3D Engine

In order to model a real world environment, the data used to represent and describe this world must be stored in three dimensional space. The problem that then arises is that to view this world, you must first find a way to translate this 3D based data into 2D data that can be mapped directly onto a 2D image.

8.1 Basic Theory

The basic theory behind 3D engines is based around using trigonometry to translate three-dimensional world coordinates into two-dimensional coordinates that correspond directly into pixel values that can be drawn onto the screen or an image. This is achieved by tracing the line between any world point and the viewpoint. The 2D screen coordinate is calculated by finding where this line intersects a given plane (the screen or image plane). See Fig 8.1.1 for a diagrammatic representation.

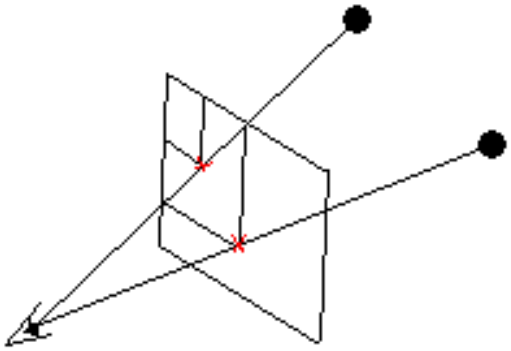


Figure 8.1.1. A depiction of the basic 3D engine theory.

Translating from 3D to 2D is not all that must be considered. The 3D description of the world itself must also be moved or rotated to bring life to your scenes and also to enable the viewpoint to be changed in both position and direction. The basic steps that have to be followed are described as “The Graphics Pipeline”.

8.2 The graphics pipeline

The graphics pipeline is so called because it is a series of stages where the output of one stage becomes the input to the next. The basic stages are shown in fig 8.2.1.

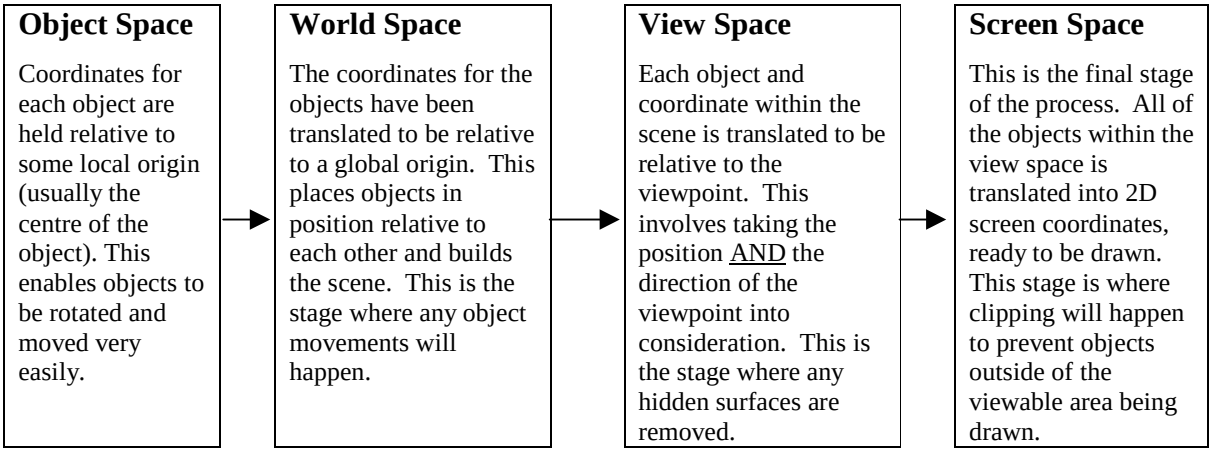

Object Space
Coordinates for each object are held relative to some local origin (usually the centre of the object). This enables objects to be rotated and moved very easily.
World Space
The coordinates for the objects have been translated to be relative to a global origin. This places objects in position relative to each other and builds the scene. This is the stage where any object movements will happen.
View Space
Each object and coordinate within the scene is translated to be relative to the viewpoint. This involves taking the position AND the direction of the viewpoint into consideration. This is the stage where any hidden surfaces are removed.
Screen Space
This is the final stage of the process. All of the objects within the view space is translated into 2D screen coordinates, ready to be drawn. This stage is where clipping will happen to prevent objects outside of the viewable area being drawn.

Figure 8.2.1. The basic graphics pipeline

The graphics pipeline will vary depending on the various features of the engine in question. For example, if the engine implements Z-Buffering (a hidden surface removal method) then there will be an extra stage between view and screen space for the "Depth Buffered Screen Space".

8.3 Major issues

You could write a whole book on the building of 3D engines, in fact many people have. Because of this I am going to concentrate on the major issues that affect the building of a 3D engine.

8.3.1 Accuracy v Speed

Blinn [1979] discusses the major concern when building a 3D engine - speed. Creating very complex 2D images from 3D data is can be a very complex and time-consuming process. The more fancy effects that are implemented, the slower the rendering time is. It is for this reason that trade offs must be made. To decide what effects should and should not be implemented, you need to look at the purpose for the engine. If you need a very fast response for real-time applications, such as a training flight simulator, it is very doubtful that you would implement texture mapping on grass and trees etc. However, if you were building an engine that has to create high quality images regardless of time taken, such as 3D modelling packages, you would probably implement every effect you could think of. A possible compromise is to use different rendering algorithms depending on the distance of the object. Becker and Max 1993

discuss the implementation of this and state the main reasons for this being that a lot of time is being wasted calculating detail that is too small to be displayed.

8.3.2 Hidden Surfaces & Clipping

Hidden surface removal is a very important part of any 3D engine. It is very important that objects in the foreground are not hidden by objects behind them. To prevent that from happening, there are a number of different methods that can be employed. The most simple is Painter's algorithm.

Painter's Algorithm involves sorting all of the polygons in a scene and drawing them in order starting with the furthest away. What this accomplishes is that the objects in the foreground are drawn on top of the objects in the background. That on it's own is not quite enough. There will be instances where the depth of polygons will overlap, in this circumstance it is not obvious which polygon to draw on top, or, if there is a possible way to draw them in order to produce the correct result.

Another method for removing hidden surfaces is the Z-Buffer method. What this means is that when you are drawing the image, an extra value is stored for each pixel drawn, when a pixel would be drawn over a pixel that has been drawn already, the depths of these pixels are compared. The new pixel will only be successfully drawn if the depth is less than that of the old pixel. This solves the overlapping problem with Painter's algorithm by not being concerned with the polygon as a whole, however, the Z-Buffer is far more complex to implement and is on the whole more time consuming.

Whichever method used, back facing polygons can be removed from the scene altogether. The most popular and easy way to remove back facing polygons is to store the vertices of the polygons in clockwise order, then checking the order of the vertices as they would appear on screen. If the vertices are anti-clockwise on the screen then the polygon must be facing away from the viewpoint and can be removed from the scene.

8.3.3 Shading / Texturing

A large part of building a 3D engine is the implementation of shading and texturing. However, I have discussed various methods previously in this report, so, rather than repeat myself, please refer to Chapter 5 - Research: Map Texture on page 22.

9. Research: Golf Course Design

Although it is not an important aspect with regards the physical realism of this golf simulation, I felt that I should investigate into the area of golf course design. One of my main reasons for this is that a lot of users will not see the complexity of the implementation of the physics, all they will see is a game, and with a badly constructed golf course, probably a pretty poor game.

9.1 Greens

I found various pointers as to how to construct the putting greens. These are as follows:

- The green should be an irregular shape.
- The green should be of sufficient size that a reasonably accurate ball struck from a fair distance away has a good chance of stopping on it. The average recommended size for a green is $500m^2$.
- The green should be contoured in such a way as to be free from moisture collecting hollows [puddles].
- The slope at any part of the green should be no more than 1 in 8.

I also discovered some guidelines to help in the building of the area surrounding the green. These are as follows:

- The area should include informal banks in a variety of shapes mounds and hollows.
- These banks should have a slope of no greater than 1 in 4.

9.2 Bunkers

There are a few general guidelines for the creation of Bunkers (Sand Traps). These are as follows:

- Bunkers should all vary widely in area, plan and elevation.
- The average size of a bunker should be about $75m^2$.
- Sand faces should vary between 30^o and 40^o from the horizontal.

9.3 Teeing Grounds

The main points to consider when designing the teeing grounds are:

- The teeing grounds should be perfectly flat, or, if it must be sloped, it should be sloped slightly uphill.
- The teeing ground should be generally rectangular and aligned roughly in the direction of the hole.
- The front edge of the teeing ground should be 300mm above the level of existing ground in front of the tee.

10. Research: User Interface

The user interface is a hugely important part of most applications, especially games. No matter how good the application is behind the scenes, users will judge it by its interface. A good interface will keep the users happy and hopefully they will come back for more.

10.1 Theory

Maddix [1991] states that HCI (Human Computer Interface) can be seen very much as an act of communication, the aim being the successful transfer of meaningful information between the user and the system.

It is very important to keep this communication as simple and effortless as possible. For example, for the user to tell the system to hit the ball, it is far more desirable to simply click the mouse button on a button labelled “Hit Ball” than for the user to manually type “Hit Ball”.

The ordering of communication is also very important, it should be as natural as possible. For example:

Good Order	**Bad Order**
Display overhead view of course	Select shot spin
Select club	Select shot power
Select shot direction	Hit Ball
Select shot power	Select shot direction
Select shot spin	Select club
Hit Ball	Display overhead view of course

10.2 Possible GUI design

I have created a design for the user interface that is relatively simplistic yet effective. Figures 10.2.1 and 10.2.2 (overleaf) show this interface. As you can see, the input makes use of scroll bars to select the required values. Whilst this is unusual for a golf game, it is actually a very standard way of selecting a value from a range and would be very effective for this situation. The direction indicator is altered simply by clicking anywhere on the screen.

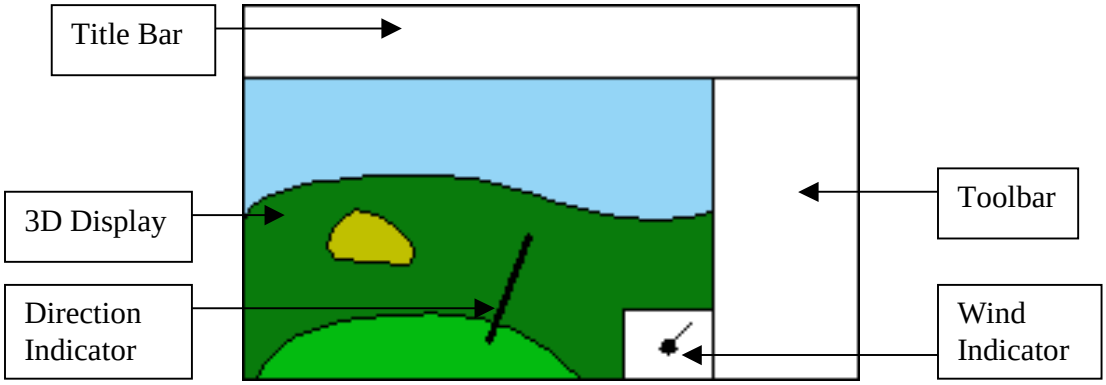


Figure 10.2.1. Possible design for the user interface

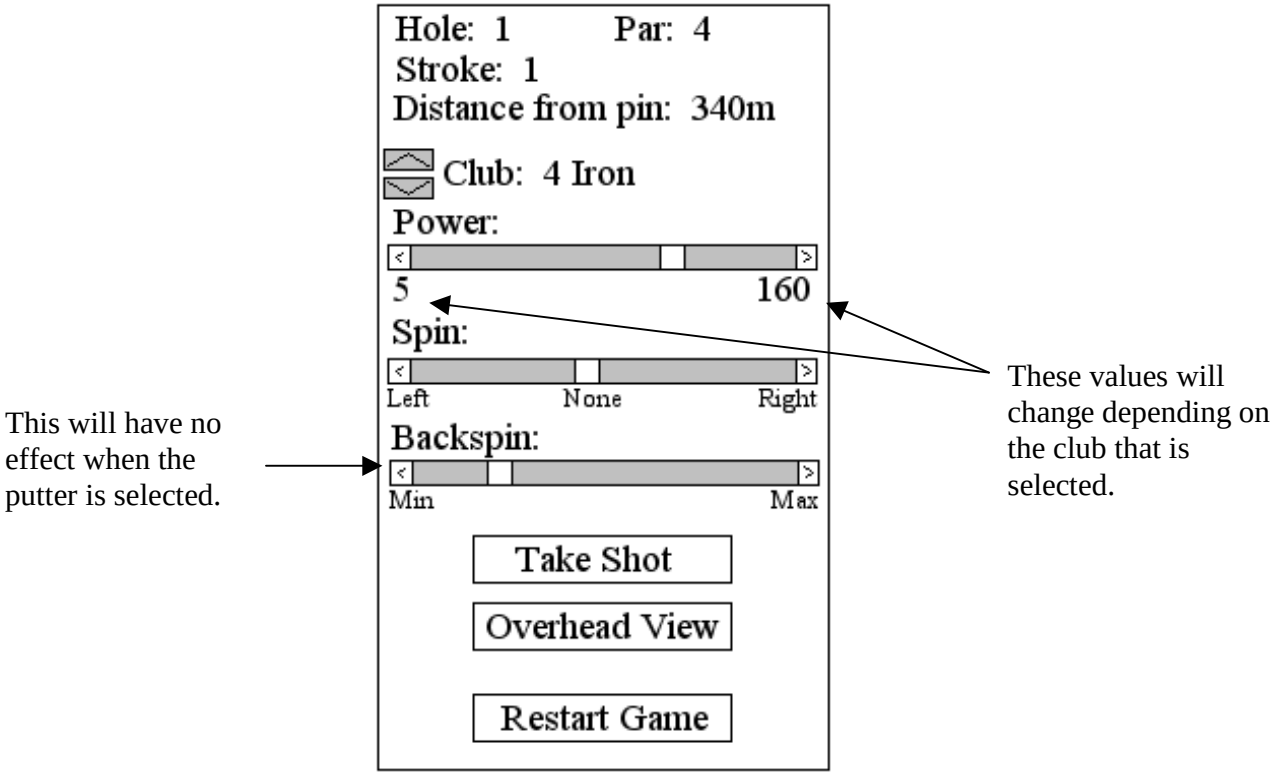


Figure 10.2.2. A possible tool bar for the user interface

11. Requirements

This chapter defines the requirements for the development of the Golf Simulation software. The requirements have been split into various sections to enable easy reading.
I decided to express the requirements in plain text rather than through some kind of formal specification language as the majority of them are observational, also the data structures are extremely complex and would be best created in the design phase.

The titles for the different sections here do not adhere to the standard expressed throughout the rest of this document. For example, the first section would have been 11.1 but is actually named R1.1. This is because the requirements should have their own coding system to enable easy referencing.

R1 Ball Travel

R1.1 Ball Flight

Code	Description	Priority	Notes
1.1.1	The flight of the ball must logically and visibly be affected by gravity, air resistance, wind and spin.	4	
1.1.2	If the ball has topspin, it will cause a downward force that will cause the ball to dip.	4	
1.1.3	If the ball has backspin, it will cause a lift force that will cause the ball to float further.	4	
1.1.4	If the ball has right spin, it will cause the ball to curve to its left.	5	Right spin is as if the ball was struck on its right side.
1.1.5	If the ball has left spin, it will cause the ball to curve to its right.	5	Left spin is as if the ball was struck on its left side.
1.1.6	While the ball is in the air, gravity will exert a force in a downward direction.	5	
1.1.7	Air resistance will result in a drag force that will cause the ball to decelerate.	3	
1.1.8	Air resistance will cause the spin on the ball to decelerate.	4	
1.1.9	Wind will cause a force on the ball with direction and magnitude being calculated from the direction and velocity of the wind **relative** to the direction and velocity of the	4	

	ball (Not relative to a stationary point).		
1.1.10	Wind will affect the magnitude and direction of the forces caused by spin on the ball.	3	See 1.1.2 to 1.1.5

R1.2 Ball Impact with Ground

Code	Description	Priority	Notes
1.2.1	When the ball contacts the ground, the ground absorbs some of the energy of the ball. The amount of energy absorbed depends on the angle of impact, the velocity of the ball and, most importantly, the texture of the ground.	5	Without this, the ball will just bounce forever.
1.2.2	If the ball has sufficient energy after the initial impact, it will bounce, otherwise, it will begin to roll.	5	
1.2.3	The spin on the ball will cause a force on the ball as in the following table: Type of spin / Direction of force Top / Forward Back / Backward Right / Very Slight Right Left / Very Slight Left Top & Right / Forward & Right Top & Left / Forward & Left Back & Right / Backward & Right Back & Left / Backward & Left	3	
1.2.4	The angle of the bounce will depend on the angle of impact, the angle / slope of the ground at the point of impact and the texture of the ground.	5	

R1.3 Ball Roll

Code	Description	Priority	Notes
1.3.1	The rolling of the ball is affected by wind, slope (angle of ground), the texture of the ground and the spin on the ball.	4	
1.3.2	The wind has a very slight affect on the ball and this affect is calculated relative to the balls direction and velocity.	3	(See 1.1.9)
1.3.3	Spin on the ball causes forces as in the table specified in 1.2.3.	5	
1.3.4	Spin on the ball is very quickly deteriorated and becomes topspin, spinning enough to move the surface of the ball at the same speed as the ball is rolling.	5	Think of it like a wheel rolling across the ground.

1.3.5	The ground offers resistance to the ball with a force that depends on the normal reaction of the surface on the ball and the texture of the ground.	5	
1.3.6	If the ground that the ball is on is sloped, gravity will exert a force on the in the direction down the slope with magnitude relative to the gradient of the slope.	5	
1.3.7	It should be possible for the ball to momentarily stop and then move again. For example, if the ball just rolled directly up a steep slope, it may momentarily stop and then roll down the hill again.	3	

R2 Golf Course

R2.1 Map Contour

Code	Description	Priority	Notes
2.1.1	The golf course must be able to represent sloping in any direction with any gradient.	5	
2.1.2	It must be possible to calculate the y-coordinate (height) for any specified x and z coordinates (Horizontal plane).	5	
2.1.3	The contour must be of sufficient detail (e.g. high frequency of polygons) to allow smooth travelling over curved parts of the course.	4	
2.1.4	The display engine should make it possible for the user to be able to see the contour.	4	
2.1.5	The contour of any part of the course that is water should be completely flat.	3	
2.1.6	It must be possible to calculate the orientation of the slope at any point on the map.	5	
2.1.7	The course must have a boundary. Balls struck beyond that boundary must be considered out of bounds.	3	

R2.2 Map Texture

<table>
<tr><th>Code</th><th>Description</th><th>Priority</th><th>Notes</th></tr>
<tr><td>2.2.1</td><td>The golf course must be able to be composed of at least the following textures:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Rough
Fairway
Green
Sand
Water</td><td>Long Grass
Medium length grass
Short Grass
Sand (as in a bunker / sand trap)
Pond / Lake</td></tr>
</table></td><td>5</td><td></td></tr>
<tr><td>2.2.2</td><td>Rough should have the following properties:
a) It should offer high resistance to balls rolling over it.
b) It should absorb a lot of energy when the ball impacts on it.
c) It should be displayed in a different colour to other textures and dark green is recommended.
d) Striking the ball from the rough should be very inaccurate.</td><td>5</td><td></td></tr>
<tr><td>2.2.3</td><td>Fairway should have the following properties:
a) It should offer medium resistance to balls rolling over it.
b) It should absorb a medium amount of energy when the ball impacts on it.
c) It is recommended to display Fairway as a medium green colour.
d) Striking the ball from the Fairway should be relatively accurate.</td><td>5</td><td></td></tr>
<tr><td>2.2.4</td><td>Green should have the following properties:
a) It should offer little resistance to balls rolling over it.
b) It should absorb only a little energy when the ball impacts on it.
c) It is recommended to display Green as a light green colour.
d) Striking the ball from the Green should be very accurate.</td><td>5</td><td></td></tr>
<tr><td>2.2.5</td><td>Sand should have the following properties:
a) It should offer medium resistance to balls rolling over it.
b) It should absorb almost all of the balls energy on impact, so, the ball will probably not bounce.</td><td>5</td><td></td></tr>
</table>

	c) It is recommended to display Sand as a brownish yellow colour. d) Striking the ball from the Sand should be fairly accurate.		
2.2.6	Water should have the following properties: a) As soon as the ball enters the water it is considered as dead. The ball cannot roll across or bounce on water. b) Water is recommended to be a bluish colour.	5	

R3 Game

R3.1 Game Rules

The game of Golf is extremely complex with far too many rules for me to implement in this timescale. For this reason I have decided not to concern myself with the full implementation of the rules of the game, and merely to provide the basic structure.

Code	Description	Priority	Notes
3.1.1	The player may not disturb the position of the ball except for on the initial tee shot where they may place the ball anywhere within the set teeing boundary.	1	
3.1.2	The player has completed the hole when they have successfully placed the ball within the cup.	1	
3.1.3	When the ball is struck into water or out of bounds they may replay the shot from the same position or take a drop ball at the nearest legal position to the place where the ball went out of bounds.	1	

R3.2 Golf Clubs

Code	Description	Priority	Notes
3.2.1	The system should offer the user access to at least the following clubs: Driver (1-Wood) 3-Wood 5-Wood 1-Iron 2-Iron 3-Iron 4-Iron 5-Iron	5	

	6-Iron 7-Iron 8-Iron 9-Iron Pitching Wedge Sand Wedge Putter		
3.2.2	The woods should give the ball a smaller angle of elevation than the irons.	5	
3.2.3	The woods should give the ball more speed than the irons.	5	
3.2.4	The higher the value of the club, the greater the angle of elevation they produce.	5	
3.2.5	The putter should only be used when the player is on the green, and should be automatically selected for the user.	2	

R4 Display

R4.1 3D Engine

Code	Description	Priority	Notes
4.1.1	The 3D engine should allow the golf course to be viewed from any specified view position and direction.	5	
4.1.2	The contours of the map must be visible. This can be implemented by simple shading.	5	See Chapter 5 section 6.
4.1.3	The different textures of the golf course should be visible. This can be implemented by simply having different colours for the different texture types.	5	
4.1.4	The engine should generate a scene in a reasonable time. This time should be as quick as possible but must be at most 1 second.	4	

R4.2 User Interface

Code	Description	Priority	Notes
4.2.1	The user interface must be easy to understand.	5	
4.2.2	The required user input must be as simple as possible. I.e. the user should not have to hold down Ctrl+Alt+Shift+S just to take a shot.	5	
4.2.3	The user interface must be very responsive. I.e. when the user does something, it should be instantly noticeable so that the user knows their input has been accepted.	5	

4.2.4	The user must be able to select the golf club that they want to use. They must also be able to change their mind and select a different club.	5	
4.2.5	The display should let the user know how far the ball is expected to travel if it is hit at full power with the currently selected club.	3	
4.2.6	The user must be able to change the direction in which the ball will be struck. They must also be able to change their mind and select another direction.	5	
4.2.7	The user must be able to select the power with which they will hit the ball. This may be an inaccurate process.	5	
4.2.8	The user must be able to select how they will hit the ball – effectively, what spin they will put on the ball. This may be an inaccurate process.	5	
4.2.9	The user should be able to view the layout of the current hole, possibly from an overhead view with a "You are here!" marker.	3	
4.2.10	The interface should restrict the types of shot that the user can take to reasonably physically possible shots.	5	For example, the user should not be able to drive the ball 600 yards.

R5 Objects

Code	Description	Priority	Notes
5.0.1	The golf course should include various real world objects that may include bushes, trees, fences and signposts.	2	
5.0.2	The ball should interact with these real world objects in a realistic way.	2	

R6 Documentation

Code	Description	Priority	Notes
6.0.1	All source code produced should be fully commented.	5	
6.0.2	This requirements document should be kept up to date from creation to completion.	5	
6.0.3	The design documentation should include details of classes, objects, relationships between objects, data structures and	5	

	interaction between objects.		
6.0.4	A user guide should be produced which should detail how the user should run and control the game.	4	

12. Risk

As this is a relatively small project with only a single person working on it, I decided not to go too in-depth with the Risk strategies. However, the considerations that I have made have been listed below.

Identified Risk:

Overrunning the allowed time.

Aversion Strategy:

To try to regain a little time, I could drop a few of my goals. I think the main area which could be left out without too much impact is the real world objects (bushes and trees etc.). I could also change my plan from creating a complete game into just a single hole demo.

Identified Risk:

Overrunning the allowed word count.

Aversion Strategy:

This is a problem that is difficult to remedy. The main thing that can be done is to put the sections of little importance into an Appendix. This is because the contents of the appendices are not added to the word count. This is still undesirable because it is doubtful that the appendices will be read as an integral part of the report and marks could still be lost.

Identified Risk:

Being unable to solve a particular programming problem.
(also known as “Hitting a Brick Wall”)

Aversion Strategy:

It is an almost certainty that during this project I will encounter a problem where the solution is not immediately obvious, or encounter a bug where the reason is not immediately obvious. In these situations I must not waste too much time on them. I think that I should set a 2 hour limit on fixing any single problem and then I should go and work on something else. Finding something else to do will not be a problem as I am going to have to be working on at least 3 or 4 things at a time throughout the running of this project.

13. Map Modelling and Representation

I have decided to make a small change to my original plan. I was going to implement the map representation within the same tester application as the Ball Travel. I have now decided that it would be better to develop the map representation techniques as an application in it's own right. This has simplified things somewhat and I can now develop this application much more rapidly without worrying about it fitting in with everything else just yet.

13.1 Map Representation

As stated in the research section, I decided to represent the map as a structured polygonal mesh. The structure I decided to use is shown in Fig 13.1.1. To store the necessary information, I decided to store all of the points in a single array. The reason that I store the points rather than polygons is to prevent repetition of data. This would be because a single point can be used by up to 6 different polygons. Storing them only once means that they can all be translated from world coordinates to view coordinates and then to screen coordinates in one simple pass.

The formula to calculate the array element representing a specified point is as follows:

```
currentPoint = ( COLS * j ) + i;
```

where COLS is the number of columns of points that exist in the map,
j = the row that the point is on,
i = the column that the point is on.

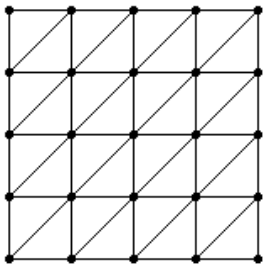


Figure 13.1.1. The chosen structure of the map.

13.2 Map Display

Fig 13.2.1 shows version 0.1a of my map representation program. It is a wire frame representation of a randomly generated map. This program can be interacted with – moving the mouse over the applet will move the viewpoint up, down, left or right, right-clicking moves forward and left clicking moves backwards. I decided to colour the lines depending on the distance from the front left. This is to allow the viewer to distinguish between the lines and is of no other significance – the reason I mention this here is because every time I demonstrate this to anyone it is the first question they ask! Fig 13.2.2 shows version 0.1b. This version is very similar to v0.1a but with a few differences. Most obviously, the polygons are filled rather than wire frame, but also, the structure represents a 3D sine wave rather than simply being random. The main change needed to go from wire frame to filled was that the drawing order had to be correct. This means drawing the polygons from the back to the front (using a simplified Painter's algorithm).

The formula to create this 3D sinewave effect is as follows:

```
pointHeight = (25*(Math.sin(Math.sqrt((j*j*WIDTH*WIDTH) +
(i*i*WIDTH*WIDTH))/60)));
```

This dissects as follows:

```
(j*j*WIDTH*WIDTH) = distance in z direction squares = z²
(i*i*WIDTH*WIDTH) = distance in x direction squares = x²

Math.sqrt(z²+x²) = distance of the point from the origin = dist

Math.sin(dist / 100) = a value between –1 and 1 that creates
the wave effect over distance = initialHeight

PointHeight = 25 * initialHeight = exaggerated sine wave =
higher waves.
```

Although the finished golf course is not going to be a 3D sinewave, this way of creating the contour with formulae will be very useful when it comes to creating the actual course.

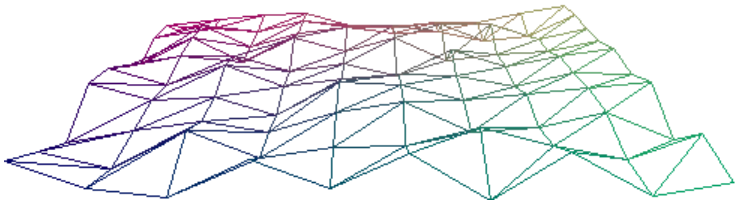


Figure 13.2.1. Screenshot of Map Representation v0.1a.

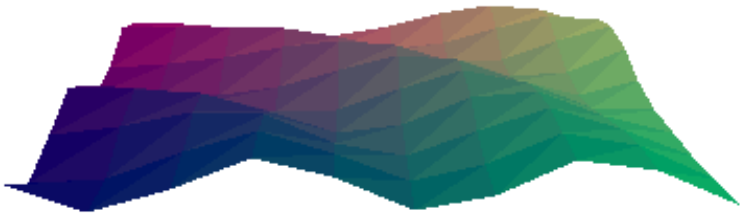


Figure 13.2.2. Screenshot of Map Representation v0.1b.

Fig 13.2.3 shows version 0.2. There is one major difference in this version. This is that the colour of the polygons is now dependant on the angle that they are facing. To accomplish this:

Calculate the normal to the plane of the polygon.

Use the Dot Product formula to calculate the angle between this normal vector and a specified light source vector.

Use this angle to define a ratio (or a percentage if you like) of the amount of green that the assigned colour will have.

What is instantly noticeable is that the curves are very jagged and the difference between the colour of the polygons is very distinct and noticeable. Fig 13.2.4 shows version 0.3, this version is actually exactly the same as the previous version except that it looks extremely smooth. All that I have done is to increase the amount of polygons and reduce the size. In fact there are exactly 19602 polygons being drawn. This screenshot demonstrates my light shading algorithm very well. In fact when I first saw the results in this version I was shocked myself!

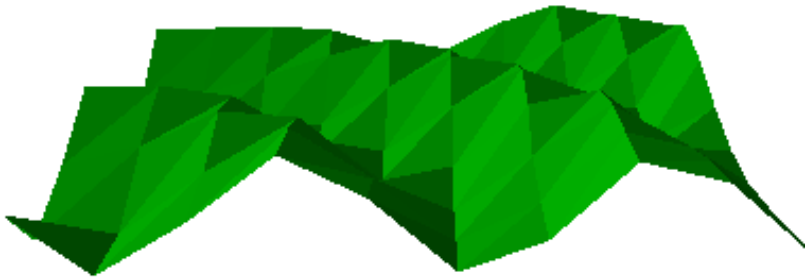


Figure 13.2.3. Screenshot of Map Representation v0.2.

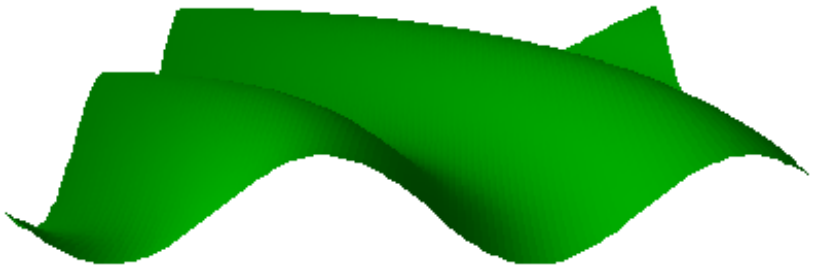


Figure 13.2.4. Screenshot of Map Representation v0.3.

Fig 13.2.5 shows version 0.4. There are significant changes between this one and the last. The main effect of these changes is that the viewpoint angle is now changeable. The user is able to look left, right, up and down as well as being able to move forwards, back and side to side. There were several steps needed to accomplish this. The major ones were:

Implement a Matrix class.

Implement matrix utility functions, such as multiplyMatrix(Matrix, Matrix).

Create matrices to take care of the rotating of the points.

Multiply each point by the translation Matrix.

The issue now is that what has always been the back of the picture can now be the front of the picture. This issue has serious implications. Mainly that the foreground may get overwritten by the background. To solve this I took the following steps:

Implement a binary tree to hold references to three points (i.e. a node will represent a triangle)

Add all visible triangles to this tree, with the tree sorting by the distances of the triangle from the viewpoint.

To draw the map, traverse the tree drawing each triangle in turn, starting with the furthest away.

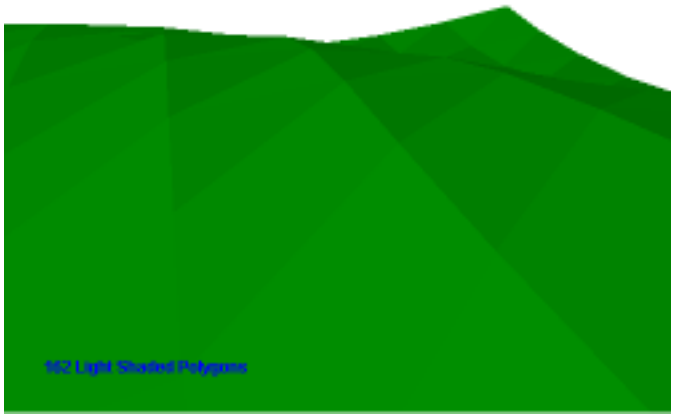


Figure 13.2.5. Screenshot of Map Representation v0.4.

Fig 13.2.6 shows a screen shot taken from version 0.5 or 0.6. These 2 versions look the same, the only difference really is the control method. The main advance over version 0.4 is the eradication of bugs. Version 0.4 was painfully slow. I initially thought it was because of the introduction of matrixes for changing the viewing angle and for the tree to sort every polygon. However, I was not fully convinced and set out to try to speed it up. I discovered that I had overlooked something – I was never clearing out the polygon tree, and so I was just permanently adding nodes. Once I stopped this, the speed was drastically improved so I made the map slightly bigger as well. The change in control method means that the user can now explore the map, use the mouse to look where they like and the arrow keys to move in the direction they are looking.

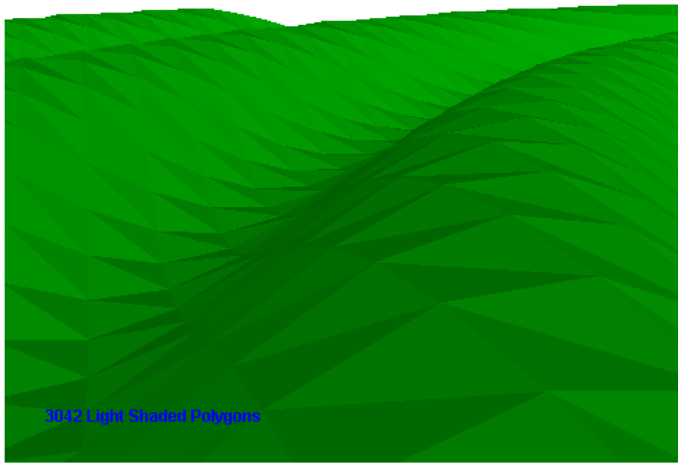


Figure 13.2.6. Screenshot of Map Representation v0.6.

The final version of this tester application that I developed is version 0.7. Fig 13.2.7 shows a screenshot taken from it. There are 2 obvious major advances in this version. The first is the introduction of different textures to the map. Shown here is Fairway, Green and Sand. The texture is allocated to different polygons mathematically, meaning that any polygons within a certain area get assigned a set texture. The main advantage of this is that I could easily raise the frequency of the polygons to produce a really smooth looking map, without any of the obviously jagged edges. The second main advance in functionality is the introduction of multiple objects. OK, the objects I created aren't exactly very useful for the Golf Simulation but they do demonstrate the power of the engine. If I do end up creating real world objects such as bushes and trees, they will be relatively easy to add into the 3D engine.

Figure 13.2.7. Screenshot of Map Representation v0.7.

All of the versions of this application have been written as java applets and are available for all to see at:
http://www.madsoftware.co.uk/
please take a look.

14. Ball Travel Algorithms

The development of the algorithms needed to realistically model the flight of the ball is one of the most complex parts of this project. For this reason I have created a separate application as a test bed for these algorithms.

14.1 Data Structure

The data structures I have designed to enable the implementation of these algorithms is shown in fig 14.1.1. The way that the different affecting factors are accounted for is by having a separate class for each, these being the Drag, Swerve and Gravity classes. The affect method of these classes returns a Force object, all the forces returned are combined to discover the resultant force on the ball. The actual equations involved are those defined in chapter 6 (Research: Ball Flight).

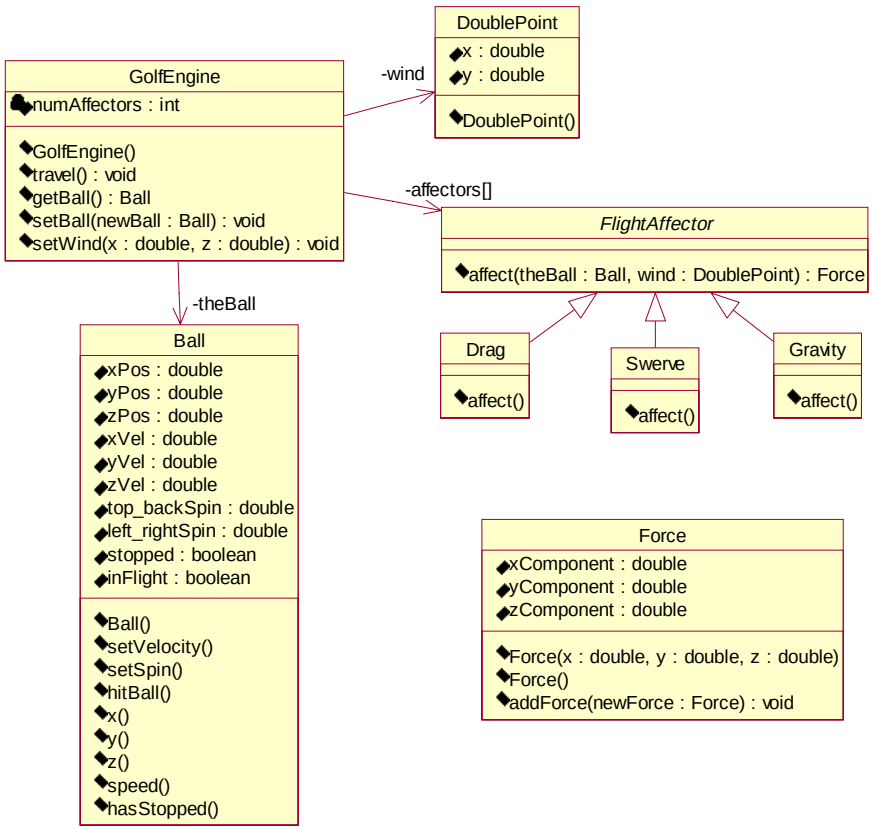


Figure 14.1.1. The data structures for the implementation of the Ball Flight algorithms.

14.2 The Application

Fig 14.2.1 shows a screenshot taken from the tester application. As you can see, it has been built to enable me to view the results from many different test runs at the same time. This has enabled me to compare the various affecting factors in order to find suitable and acceptable value ranges.

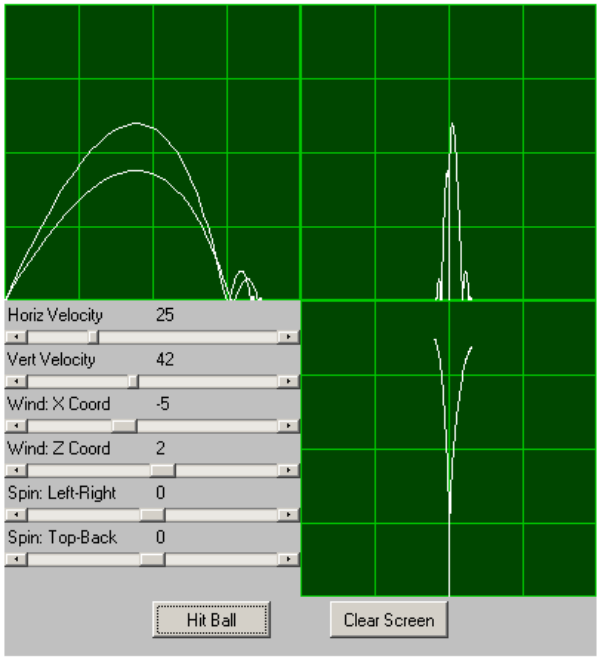


Figure 14.2.1. Screenshot of the "Ball Flight Tester" application.

This application was originally going to be built with a 3D view, but I decided to build the map view separately so, as you can see, I placed the control panel in its place. The control panel allows you to adjust the initial velocity, the wind and the spin imparted on the ball.

15. Design

This section is the system design for the overall simulation. It includes the bringing together of the two previous sections (The map display and the ball flight). This is the only part of the report where figures may be placed and not commented on. This is because the diagrams are themselves the design and should be as explanatory as possible. Comments are included where further explanation is required.

15.1 Class and Object diagrams

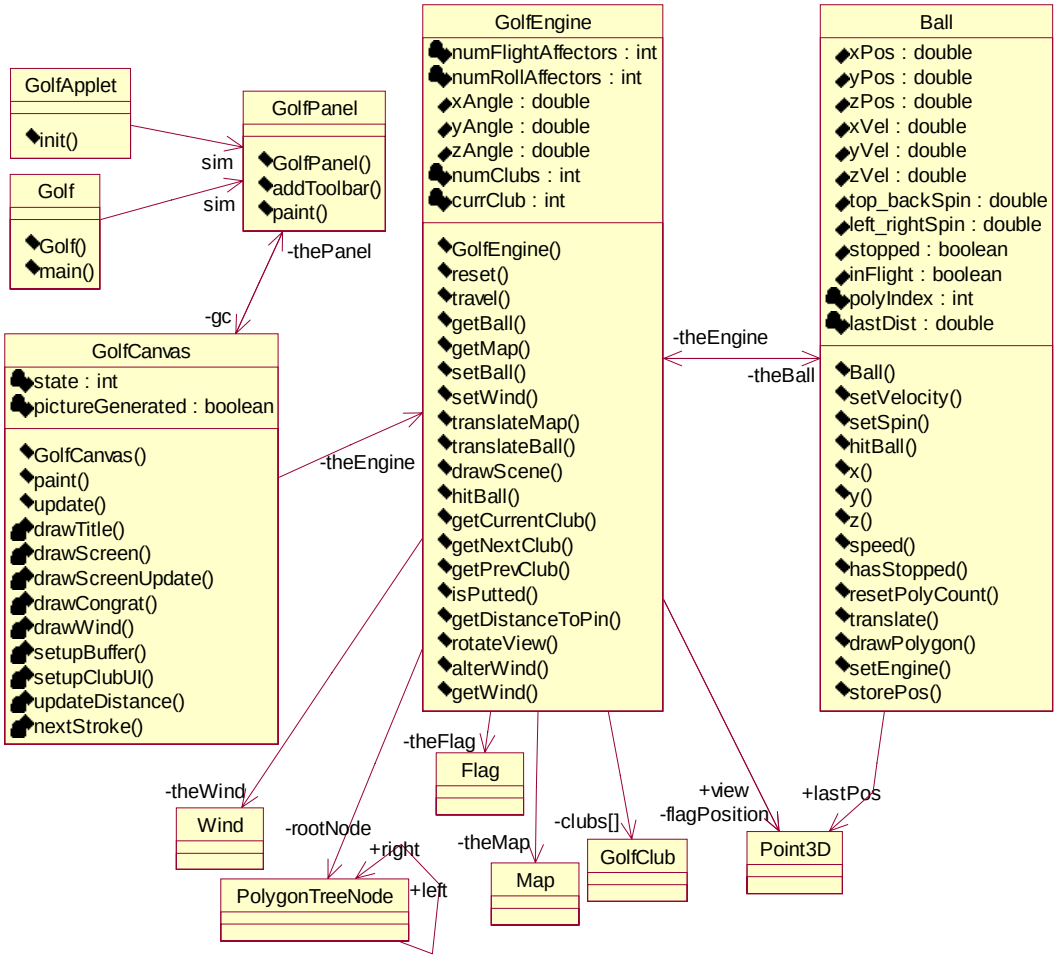


Figure 15.1.1. Class diagram showing the main structure of the system.

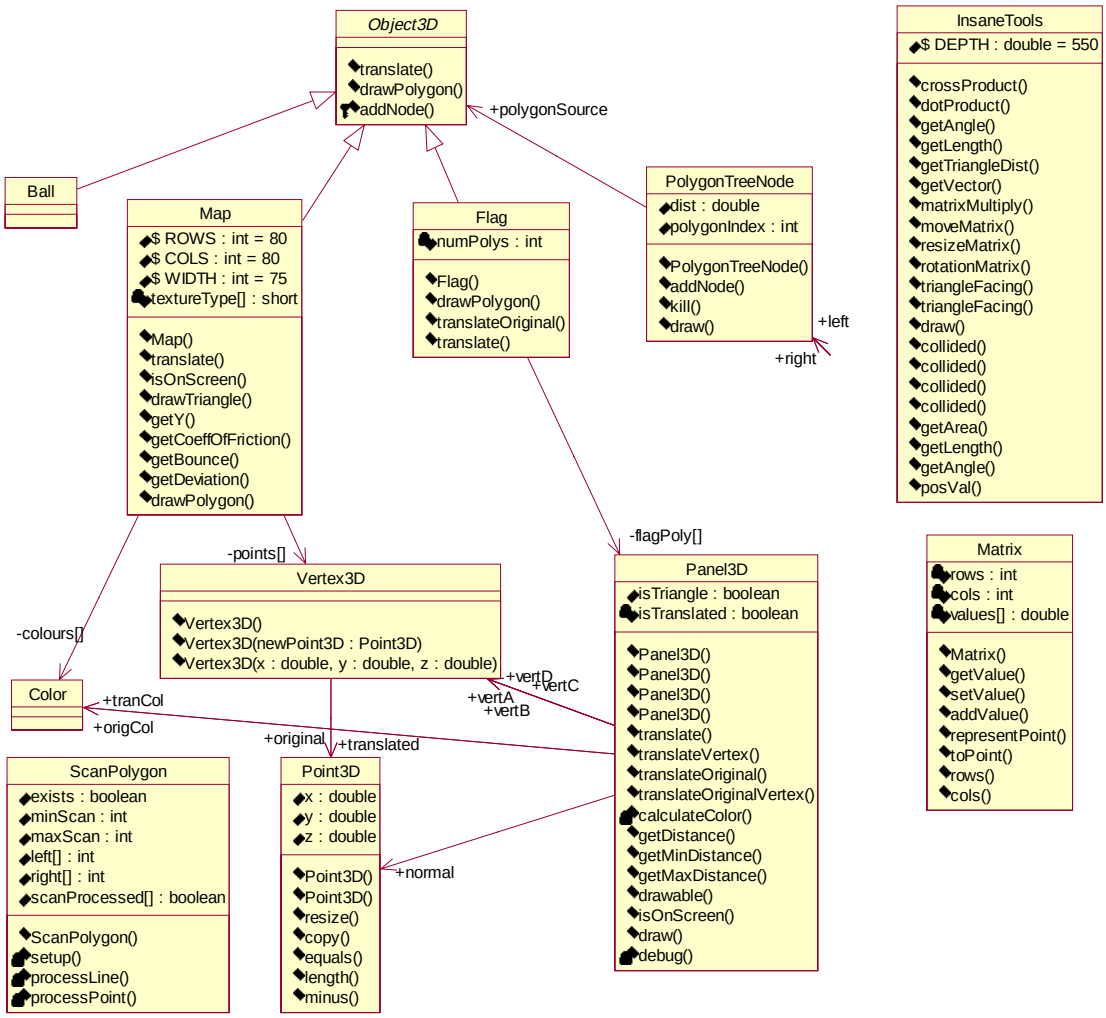


Figure 15.1.2. Class diagram showing the classes concerned with the 3D engine

NB. The Object3D class is an abstract class. This means that it can be extended, references to it can be held but it can not be instantiated.

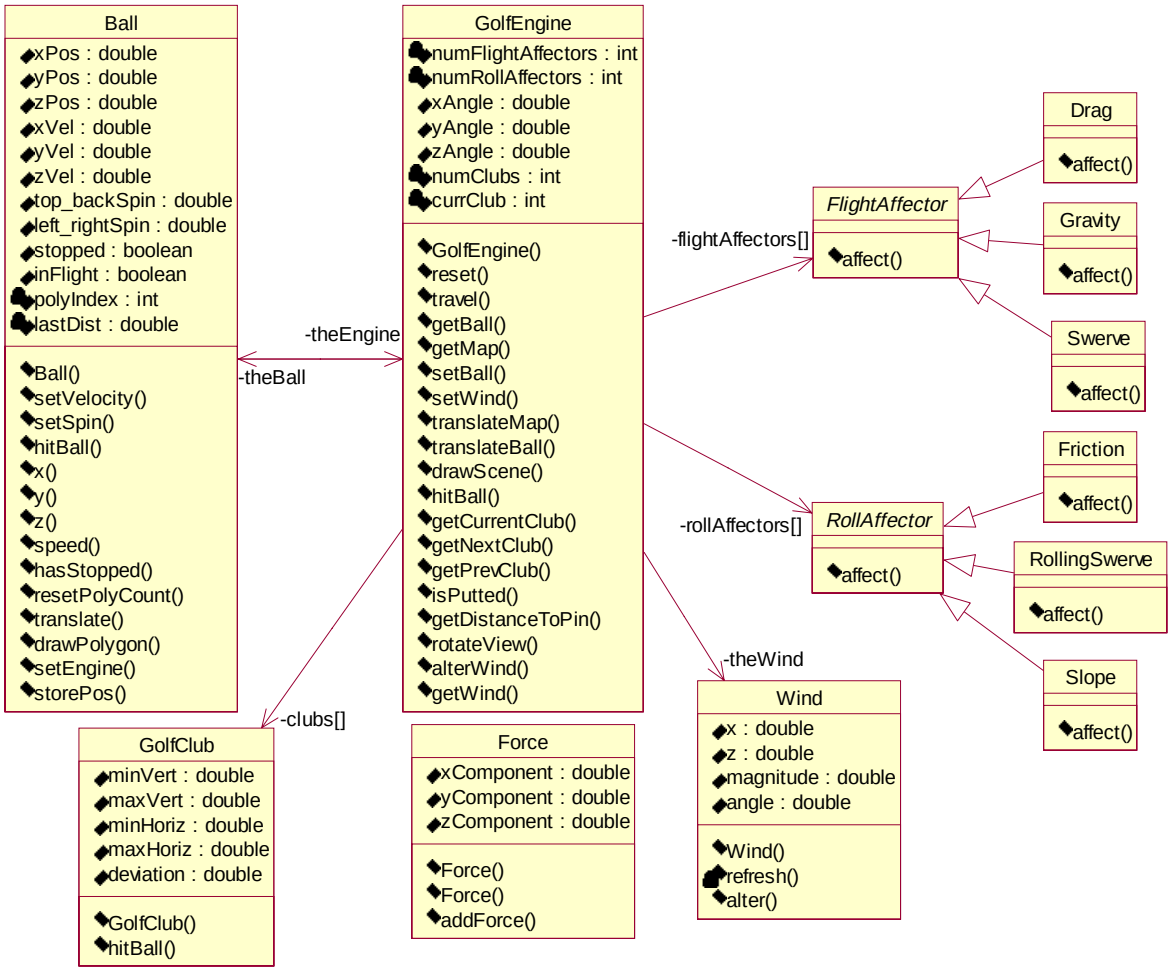


Figure 15.1.3. Class diagram showing the design for the implementation of physics.

NB. The FlightAffector and RollAffector classes are abstract classes. See above for explanation.

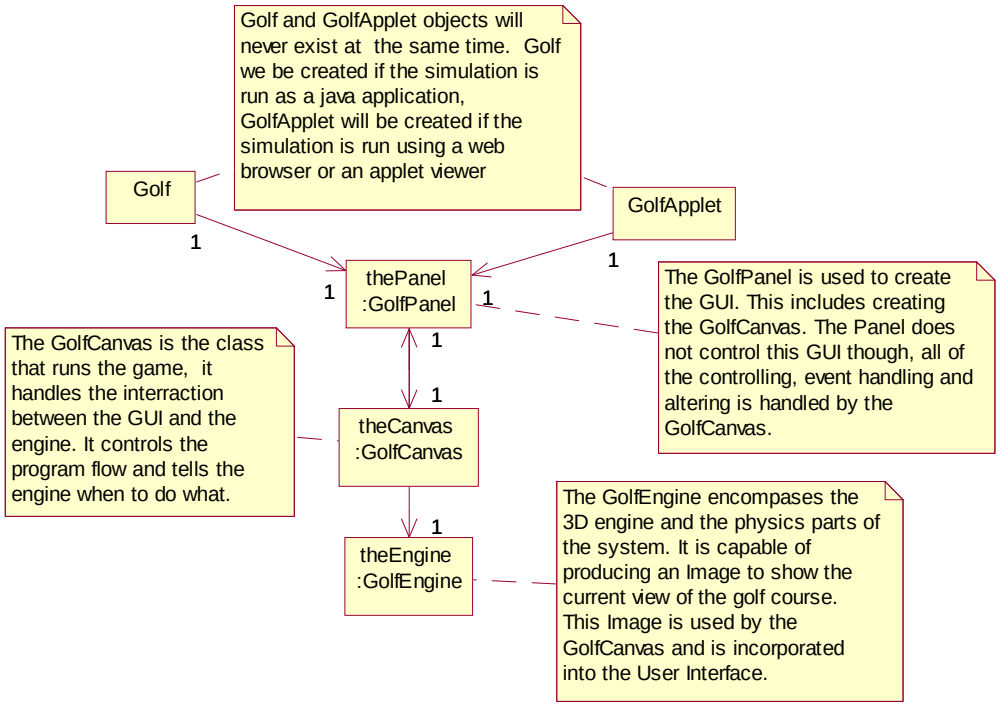

Golf and GolfApplet objects will never exist at the same time. Golf we be created if the simulation is run as a java application, GolfApplet will be created if the simulation is run using a web browser or an applet viewer
Golf
GolfApplet
1
1
1
1
thePanel
:GolfPanel
1
1
The GolfPanel is used to create the GUI. This includes creating the GolfCanvas. The Panel does not control this GUI though, all of the controlling, event handling and altering is handled by the GolfCanvas.
The GolfCanvas is the class that runs the game, it handles the interraction between the GUI and the engine. It controls the program flow and tells the engine when to do what.
theCanvas
:GolfCanvas
1
theEngine
:GolfEngine
The GolfEngine encompases the 3D engine and the physics parts of the system. It is capable of producing an Image to show the current view of the golf course. This Image is used by the GolfCanvas and is incorporated into the User Interface.

Figure 15.1.4. Object diagram showing instances of the main driving classes.

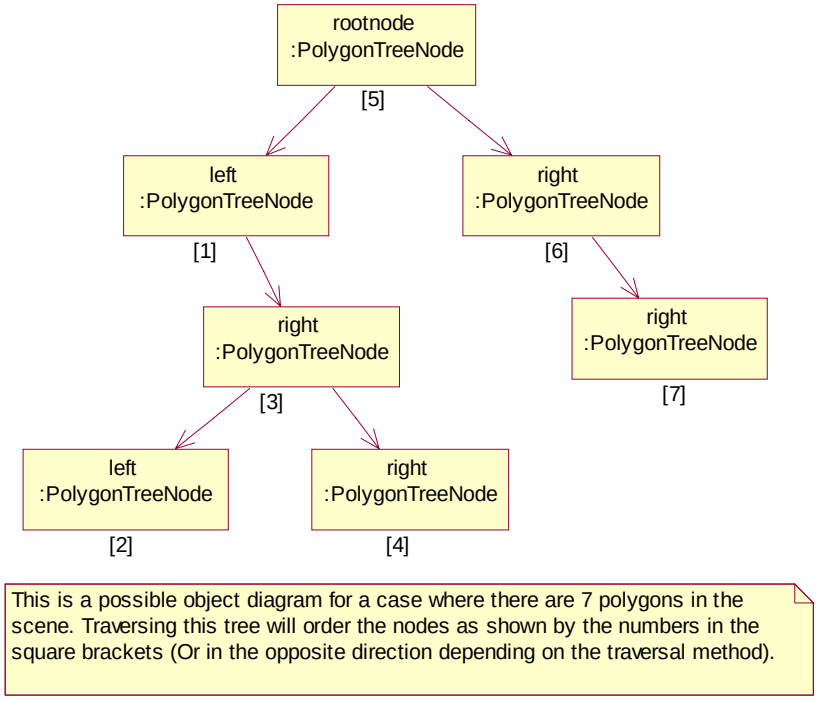


Figure 15.1.5. Object diagram showing a possible structure for the polygon tree.

15.2 State Charts

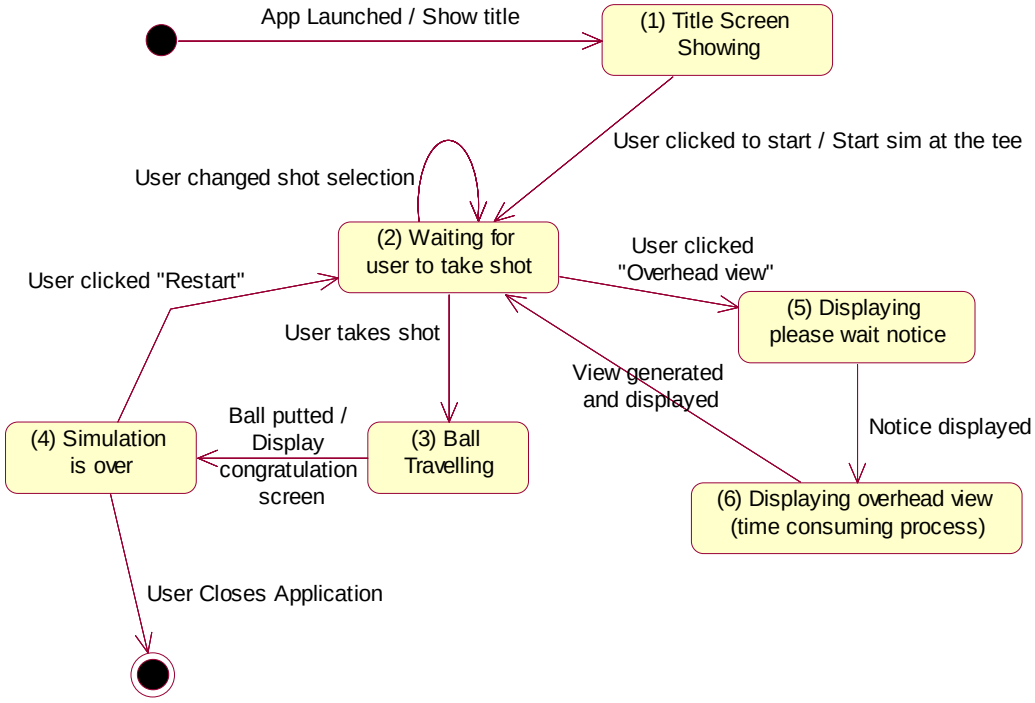

App Launched / Show title
(1) Title Screen Showing
User clicked to start / Start sim at the tee
User changed shot selection
(2) Waiting for user to take shot
User clicked "Overhead view"
User clicked "Restart"
(5) Displaying please wait notice
User takes shot
View generated and displayed
Notice displayed
(4) Simulation is over
Ball putted / Display congratulation screen
(3) Ball Travelling
(6) Displaying overhead view (time consuming process)
User Closes Application

Figure 15.2.1. State chart showing the transitions between the main states of the system.

15.3 Sequence Diagrams

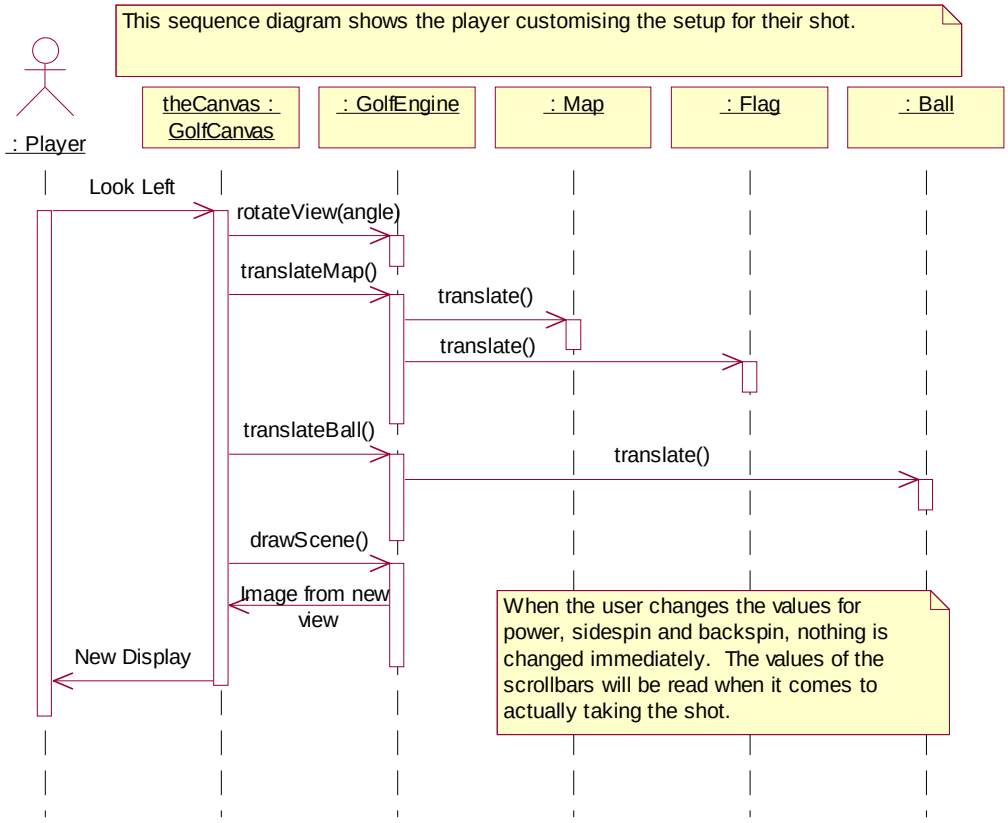


Figure 15.3.1. Sequence diagram for the player selecting their shot.

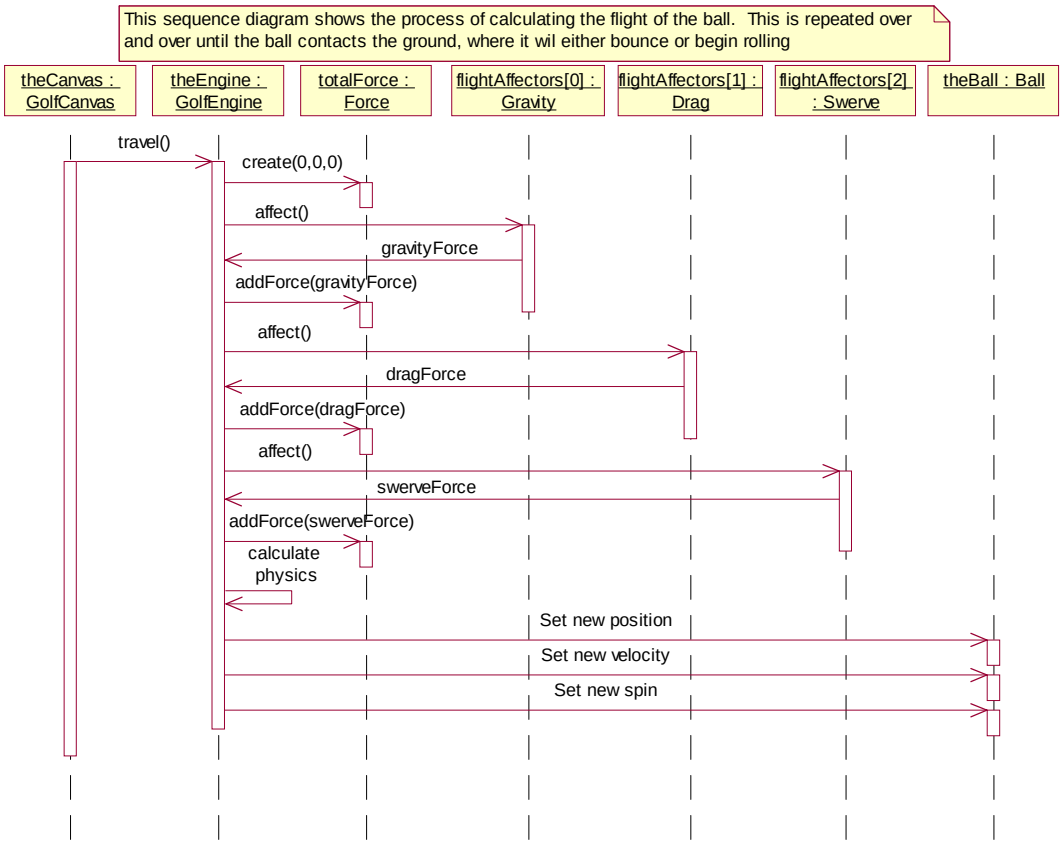


Figure 15.3.2. Sequence diagram - The calculation of physics behind the ball flight.

NB. The sequence of calculating the rolling of the ball is almost exactly the the same as the flight but with the RollAffectors (Slope, Friction and RollingSwerve) instead of the FlightAffectors (Gravity, Drag and Swerve).

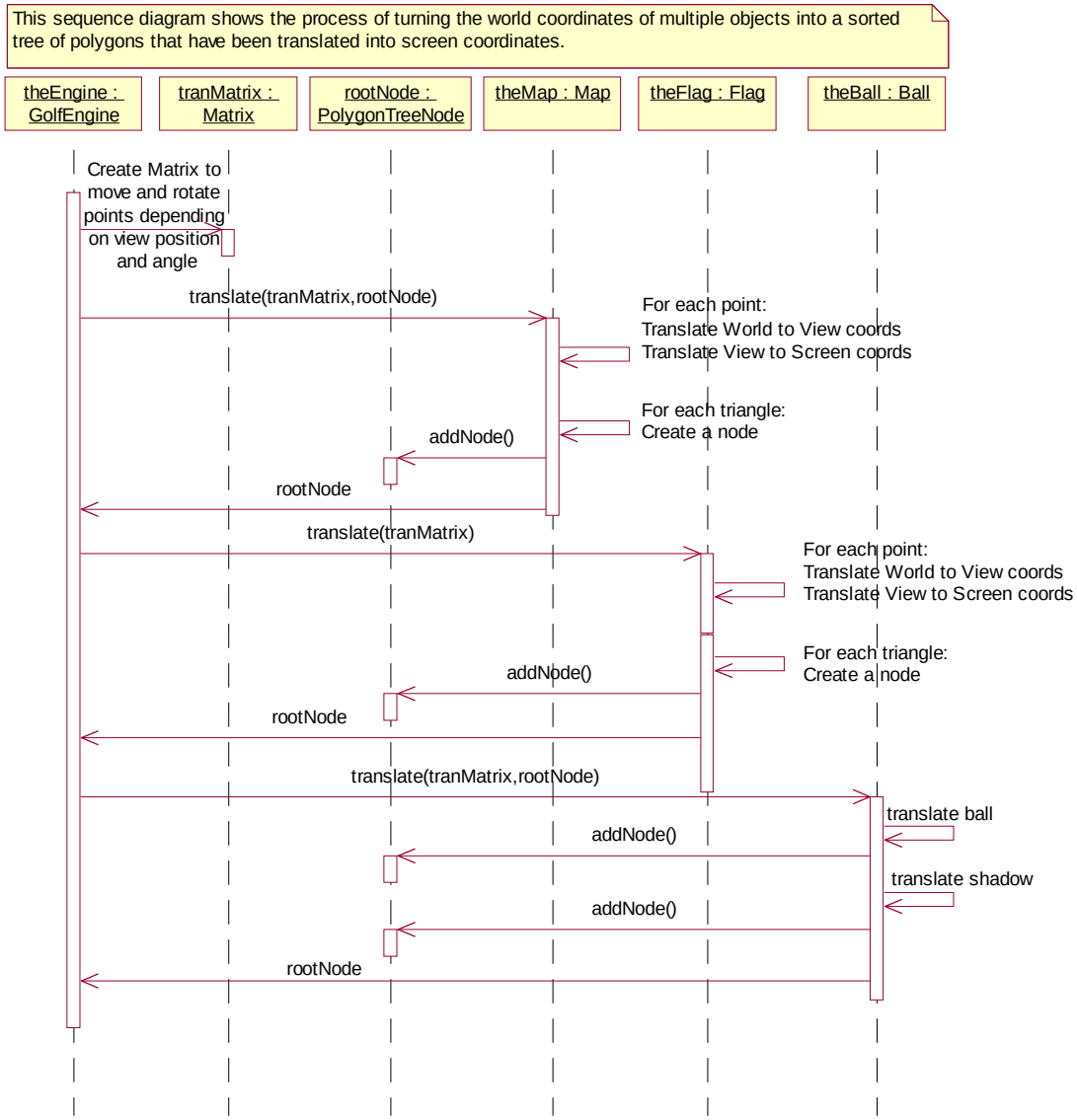


Figure 15.3.3. Sequence Diagram showing the 3D scene translation.

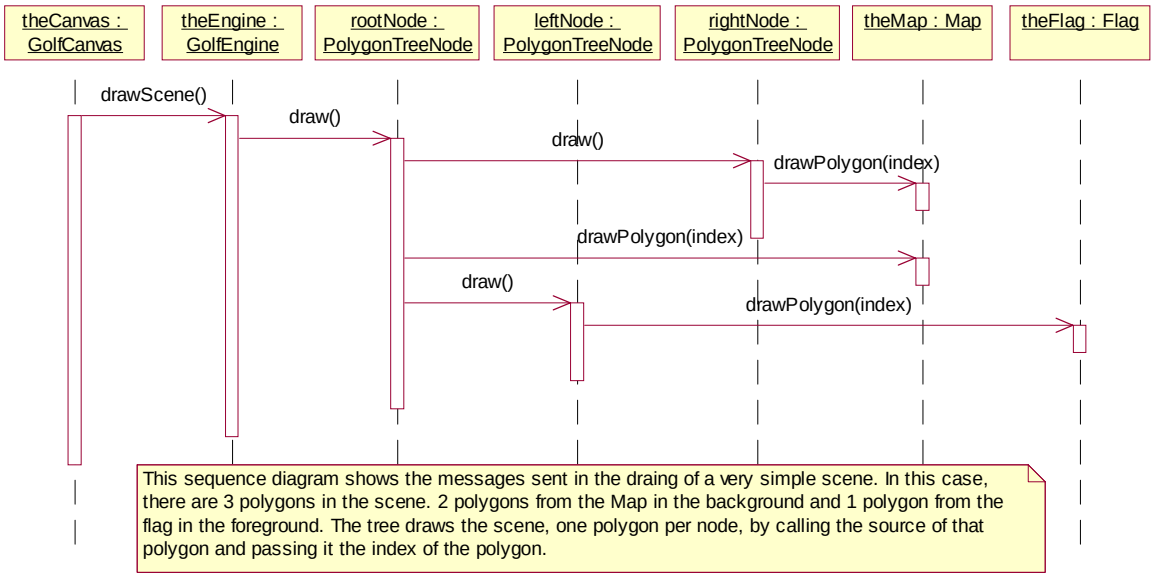

theCanvas : GolfCanvas
theEngine : GolfEngine
rootNode : PolygonTreeNode
leftNode : PolygonTreeNode
rightNode : PolygonTreeNode
theMap : Map
theFlag : Flag
drawScene()
draw()
draw()
drawPolygon(index)
drawPolygon(index)
draw()
drawPolygon(index)
This sequence diagram shows the messages sent in the draing of a very simple scene. In this case, there are 3 polygons in the scene. 2 polygons from the Map in the background and 1 polygon from the flag in the foreground. The tree draws the scene, one polygon per node, by calling the source of that polygon and passing it the index of the polygon.

Figure 15.3.4. Sequence diagram showing the drawing of a scene.

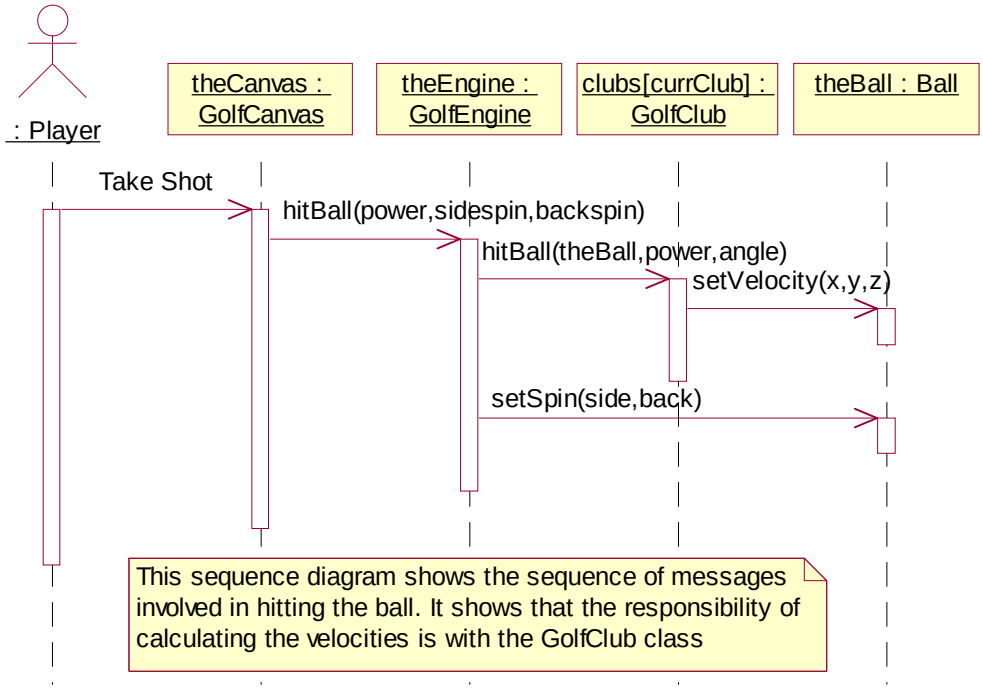

: Player
theCanvas : GolfCanvas
theEngine : GolfEngine
clubs[currClub] : GolfClub
theBall : Ball
Take Shot
hitBall(power,sidespin,backspin)
hitBall(theBall,power,angle)
setVelocity(x,y,z)
setSpin(side,back)
This sequence diagram shows the sequence of messages involved in hitting the ball. It shows that the responsibility of calculating the velocities is with the GolfClub class

Figure 15.3.5. Sequence diagram showing the interraction when the player takes their shot.

16. Test Plan

As this simulation will not be an information system as such, the types of test O can successfully perform are very limited. I cannot specify a pre planned input and check that the output is equal to a pre calculated output. What I can do is to check that various observations occur. For these reasons, I have kept this test plan to a rather simple list of tests. These tests are:

1. Does the ball fly realistically? Is it affected by gravity, wind and spin in the correct way?
2. Does the ball bounce realistically? When the ball bounces on a sloped surface does it bounce at an appropriate angle? When the ball lands on a soft surface, does it bounce lower?
3. Does the ball roll realistically? Is it affected by the slope of the ground, the wind and the spin on the ball?
4. Does the golf course look relatively realistic? Can you judge the contour and texture of the course visually?
5. Do the golf clubs react realistically? Do the irons loft the ball higher than the woods for example?
6. Is the user interface adequate? Can the user accomplish any task they wish to without too much trouble?
7. Does the application ever crash?
8. Does the application ever produce unexpected results?

17. Game Implementation

17.1 Coding Standards

In an attempt to keep the source code as readable and maintainable as possible, I have decided to draw up some simple coding standards. These are as follows:

17.1.1 Naming Conventions

All Classes, methods, attributes and local variables must be named in such a way as their meaning is obvious. I do not believe it is necessary to denote type (String, int etc) and scope (Global, local etc) in the name of variables / attributes as this makes their names far more complex and difficult to follow.

17.1.2 Requirements Tracing

To help with tracing the implementation of the requirements, I am going to add comments into the source code wherever the code is directly or indirectly satisfying a requirement. The comments will be in the following format:

//REQ 1.3.2

This standard format must be followed so you can search the text for any requirement.

17.1.3 Commenting

All source files should be commented with a header, stating when it was created and what the purpose of the class is.

Where necessary, comment within the source code to help explain the purpose and theory behind the code.

The closing curly brace " } " of each method should be commented with the name and parameter list of that method. For example:

```
}//drawScene(Graphics)
```

This is to enable easy navigation around the source file.

17.2 Screenshots

The following images are actual screenshots taken from the finished simulation. As this simulation was written so that it could be displayed as a java applet, you can play this simulation online at http://www.madsoftware.co.uk/

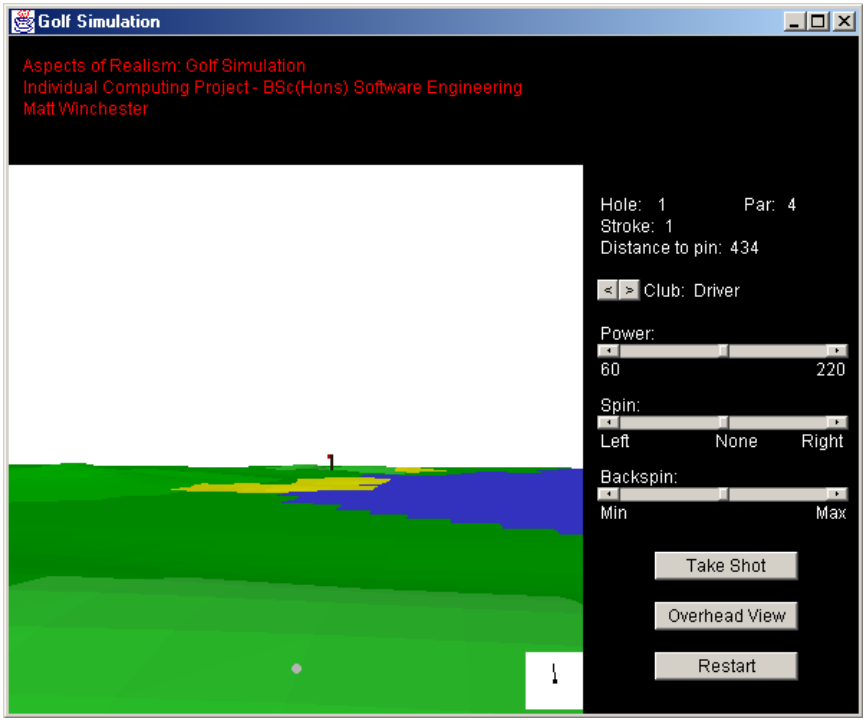


Figure 17.2.1. Screenshot: The teeing ground, looking towards the pin.

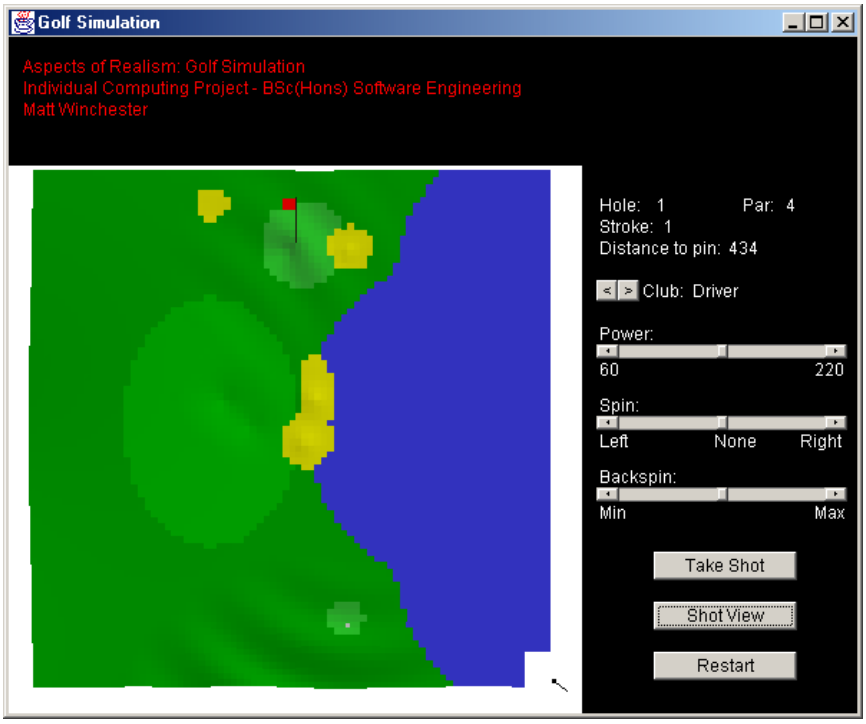


Figure 17.2.2. Screenshot: Overhead view. Notice the ball on the teeing ground.

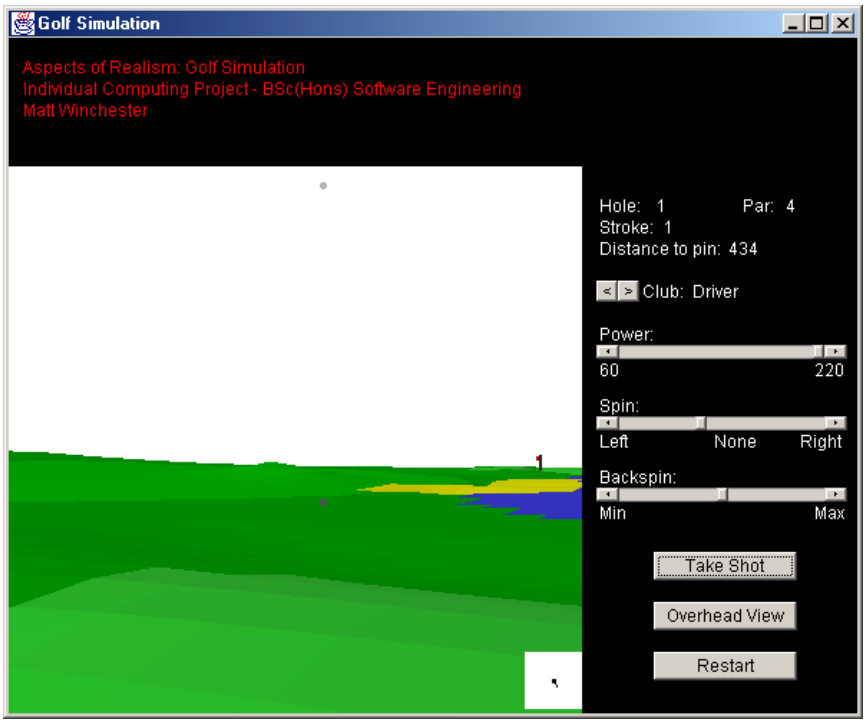


Figure 17.2.3. Screenshot: The ball in flight, notice the shadow approaching the fairway.

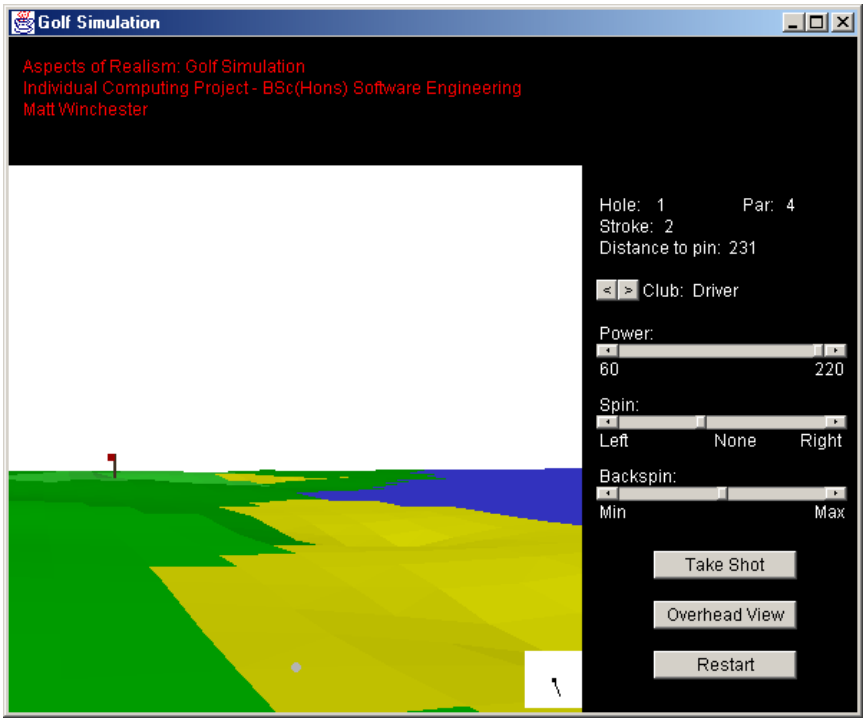


Figure 17.2.4. Screenshot: The ball in a bunker.

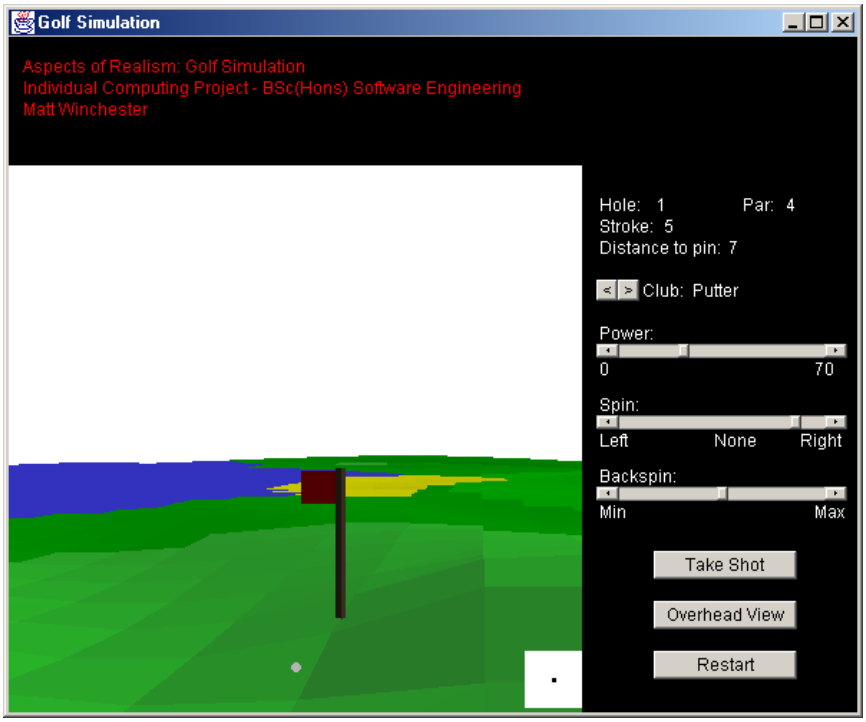


Figure 17.2.5. Screenshot: The ball on the green, 7m from the pin.

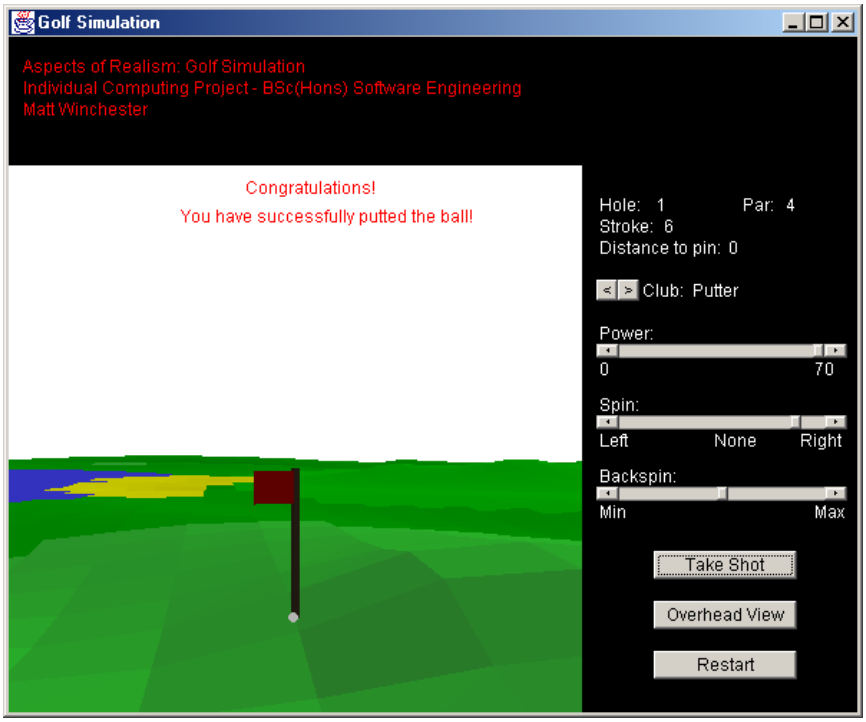


Figure 17.2.6. Screenshot: The ball was putted.

17.3 Skipped Requirements

The vast majority of the requirements have been satisfied. However, a few have not. These are detailed below along with the reason for skipping them.

R2.1.7 The course must have a boundary. Balls struck beyond that boundary must be considered out of bounds.

I have not implemented this requirement purely due to lack of time. It is not at all technically difficult and could easily be implemented in future.

R3.1.3 When the ball is struck into water or out of bounds they may replay the shot from the same position or take a drop ball at the nearest legal position to the place where the ball went out of bounds.

Again, this was not implemented due to time constraints. There is no technical difficulty in implementing this.

R3.2.5 The putter should only be used when the player is on the green, and should be automatically selected for the user.

Once again, this requirement was skipped due to time constraints rather than technical difficulty.

R5.0.1 The golf course should include various real world objects that may include bushes, trees, fences and signposts.

R5.0.2 The ball should interact with these real world objects in a realistic way.

These 2 requirements were skipped due to time constraints. The technical difficulty of these 2 requirements is fairly high. Adding the objects into the scene would not be a major problem as the engine is built to display objects (the flag for example), however, realistic interaction would have been quite a task. With extra time I would have liked to implement this.

17.4 Known Bugs

There are no bugs as such. There is one flaw with the program that I haven't had time to eradicate. This flaw is that occasionally the 3D engine will encounter a scene that takes far too long to generate. This results in the changing of direction to pause, or the flight of the ball to be extremely jerky. The reason for this is that I have not implemented full polygon clipping. What this means is that if a polygon is at all on the screen, I draw it. When the polygon has been translated, it may be just on the screen and a large part of it is off the screen. It is this off-screen drawing that is causing the major lags. To solve it all I have to do is to clip the polygons to the edges of the screen, however, I have not had time to do this, and seeing as it is not a major bug and does not crash the application, I am willing to hand it in as it is.

17.5 Results From Test Plan

Every test from the test plan was successful. The application looks and feels very natural and realistic. The user interface makes possible any shot you wish to take. The application never crashes but occasionally pauses (see known bugs above).

18. Problems and Changes

In this section I identify the areas of the project where I encountered problems and state how I solved them. Also, I identify the changes I made to the project since I created the project plan and the initial proposal.

18.1 R.U.P.

In my project proposal (Appendix A), I stated that I would be using the RUP (Rational Unified Process) throughout the lifecycle of this project. However, I changed my mind. The RUP is most useful in the development of multi-user and information centred systems, it didn't really tie in with this application. I also believe that it would have taken far more time, and with there being a shortage of time anyway, I was very quick to decide against using it.

18.2 User Interface

My initial plan completely overlooked the development of the user interface. So I just went ahead and did some research and Interface creation anyway. See Chapter 10.

18.3 Scheduling

My estimations for the time the various stages of the project were completely wrong. I underestimated the impact of coursework from other modules on my time. At one point I had been up to 2 months behind my schedule, luckily I had been generous with a few of the later tasks and also I had planned to finish by March 22nd whereas the project didn't need to be completed until May 4th.

18.4 Real World Objects

Real world objects (Bushes, Trees and Signposts for example) have not been implemented in any great depth because I was running out of time and decided to lower my aims. I have added the ability into the 3D engine for any objects to be incorporated into the world (as I have done with the flagpole) but have not had the time to create any complex objects such as bushes.

18.5 Ball Flight Animation

I encountered a very large problem when I first tried to integrate the map display with the ball flight. The problem was that I had to find some way to show the flight of the golf ball in 3D, I could simply render the flight frame by frame, however, this would be extremely slow. I could simply render the map display once and then just draw the ball on top of the static image, however, this would fail when the ball would should be obscured by the contour of the map, as it would simply be drawn over the top of everything.

Solution:

What I decided to do was relatively clever. For the entirety of the flight sequence, the viewpoint could be the same, this means that the map only has to be translated once. The polygon tree created from this is kept throughout the sequence. When the ball is repeatedly translated and added to the held tree, the index is added incrementally. What this means is that there will be many nodes in the tree representing the ball but each will have a different index. When it comes to drawing the scene, you can simply draw ONLY the latest index. Doing this means that you miss out on the time-consuming map translation phase. It also means that I could easily add in an instant replay function that shows the trace of the flight (i.e. draw all of the instances of the ball in the tree).

18.6 Ball Shadow

Another problem that I came across when implementing the ball's interaction with the map was that it was not always obvious whether the ball was rolling or flying. To solve this

problem, I gave the Ball a shadow, this shadow is unrealistic in the way that it is directly below the ball, rather than angled but it gives a very good representation of where the ball is.

18.7 The Word Count

This report has a maximum word count of 15000. I have had problems in keeping under this limit and have had to make a few compromises in order to comply with the limit. In my Risk strategy, I suggested moving some chapters into the Appendices as they don't count in the overall word count, however, I could not see any sections that I would have voluntarily placed in an Appendix. I consulted my project supervisor on this matter and he informed me that the initial project proposal didn't actually have to be part of the report, so I put that into Appendix A and wrote a far smaller introduction to give myself a bit more space.

19. Future Work

There are a lot of things that could be improved about this application. Possible future work is outlined below.

19.1 Fixing the lag problem

The problem of the application pausing occasionally is the most frustrating part of the application and is the only thing that is stopping me from being completely happy with what I have produced. If I were to continue working on this project, that would be the first thing that I would do.

19.2 Real World Objects

The inclusion of real world objects was part of my original plan and was omitted due to lack of time. This would be the next major aspect that I would work on.

19.3 Multiple players

Including the ability for multiple players would make this much more of a complete game. It is not necessary for the simulation though as the emphasis is on the physics.

19.4 Network play

Once multiple players have been implemented, I would love to make the game networkable with play over a LAN and hopefully the internet too.

19.5 Multiple holes

If I were to work on this more to make it into a proper game, I would have to build in the ability to have many different golf courses with multiple different holes.

19.6 Level editor

As the golf course layout is generated mathematically, it would be fairly trivial to build a level editor so that players could make and play their own golf courses.

20. References

Vartan Kupelian www.golfonline.com 2001

Barry G Becker, Nelson L Max “Smooth Transitions between Bump Rendering Algorithms” 1993

Neville De Mestre “The Mathematics of Projectiles in Sport” 1991

Frank Maddix “Human-Computer Interaction” 1990

Alan Watt “Third Edition 3D Computer Graphics” Addison Wesley

John De Goes “Cutting Edge 3D Game Programming with C++” Coriolis Group Books, 1996

James F. Blinn “Realism In Computer Graphics” Caltech/JPL, 1979

William E. Lorenson, Boris Yamrom “Golf Green Visualization” 1992

Fred W. Hawtree “The Golf Course: planning, design, construction and maintenance” 1983

"Friction," Microsoft® Encarta® Online Encyclopedia 2001
http://encarta.msn.com © 1997-2001 Microsoft Corporation. All rights reserved.

Kristin J. Dana, Bram Van Ginneken, Shree K. Nayar, Jan J. Koenderink “Reflectance and Texture of Real-World Surfaces” ACM Transactions V18 No 1. 1999

Appendix A: Project Proposal

1.1 Introduction

Computer simulations of the real world have a very wide array of uses. These range from entertainment and training to research and speculative planning. Examples of these are as follows:

- Games

  With the advances in hardware over the past few years, games have progressed greatly in the way of realism. It is actually quite rare for games nowadays to be released that are not 3D representations in some way. In fact the most popular type of game at the time of this project is the "First Person Shoot-em-up". These games make the player feel as if they are actually running around, jumping and shooting. Of course, with games like these, the emphasis is on making the software "look" real rather than being based on ensuring the accurate implementation of the laws of physics.

- Flight Simulations

  Flight simulations are used in the training of pilots. They are an invaluable tool because they place the trainee in a very realistic simulation of a situation where decisions and actions are critical and even life threatening.

- Advanced CAD/CAM

  There are several instances of software where simulation is used for planning and designing. An example of this is the software used in the construction of Thrust SSC (the Super Sonic Car). The software was vital in the design of this vehicle to ensure that the airflow over the surfaces at immense speeds would not cause any sort of dangerous incident such as the car lifting or flipping.

As the title suggests I am planning to create a simulation of the game of golf. To do this I am going to have to model and represent a complete 3d world. There are several distinct parts of this project that I have identified. These are as follows:

1) **Map Contour** - The logical modelling and representation of a three-dimensional contoured terrain. (The golf course)

2) **Map Texture** - The modelling of different textures of the terrain and how the textures affect the ball. Textures include short grass / green, medium grass / fairway, long grass / rough, sand and water.
3) **Objects** – The modelling and representation of objects (such as trees and bushes). I will have to model the effect they have on the ball when the ball hits them as well as finding an efficient and realistic way to display them.
4) **Ball Travel** - The modelling of the ball. This covers the flight of the ball, the effect of impact on the ball and the rolling of the ball. Factors to be taken into account include gravity, air, wind, effects of spin etc.
5) **Clubs** - The modelling of different types of golf club, the different effects they can have on the ball in terms of spin, power and direction.
6) **3D Engine** - The three-dimensional display engine. Various aspects need to be implemented, such as translation of 3d world points into 2d screen points, drawing of filled 3d polygons and rendering scenes (displaying closer objects over distant object etc.).
7) **Game Engine** - The implementation of game rules. Enabling multiple players to compete against each other. Time permitting possibly include computer opponents of variable skill levels and maybe even network or internet play.

There is also a separate small project (see "Methods" below and **Figure 1**) that I plan to design and build as follows:

8) **3D Tester** - Compromises of 3 simple 2d representations - top, front and side along with a simple point/line 3d representation.

I realise of course that there are many golf simulations in existence already. I also realise that I have very little chance of creating a game that looks anywhere near as nice as the commercial golf games. This project however offers something that the others don't – it is based around reality rather than gameplay. There are several major flaws in commercial golf games that are in place to aid in gameplay. These are factors such as wind. Wind is taken into consideration in most golf games but once you have struck the ball, the wind stays constant. Also, the majority of golf games have mostly flat courses. I have personally played on a few courses and on all of them there were holes that were either majority up hill or majority

downhill. My simulation will hopefully fill both of these gaps. I will implement wind that changes constantly and I will also ensure that I implement realistic course contours.

My main reason for taking this project is because one of my major interests is the way in which real world situations can be simulated and broken down into a series of equations. This is also where my interest in Physics comes from. This project will allow me to investigate, design and implement various ways of storing and expressing real world entities and events as objects and operations.

1.2 Aims and Objectives

The main part of this project is simulating realism. I do aim to create a very entertaining and playable game but the emphasis will be firmly on accurately representing the physics behind the game of golf rather than literal rules of the game.

A large part of this project will be to implement a 3D engine. I will need to do this so that I can display the 3D representation of the world on a 2D screen. This 3D engine will have to be able to translate points from a 3D world to 2D screen points from any viewpoint and angle. It will also have to be able to render a realistic image of the landscape (the golf course) so that the player can picture the situation they are in.

Overall, the main aim of this project is to create a piece of software that simulates the game of golf so well that someone who plays golf regularly would say “That is actually what would happen!”.

1.3 Expected Outcomes / Deliverables

At the end of this project I plan to have produced the following (not necessarily in order):

- Project documentation
  - Requirements Specification
  - Design Document
  - Source Code
  - Project write-up

- Application
  - Java classes runnable as standard application or web applet
- Additional Documentation
  - User guide explaining how to play the game.

1.4 Methods

Throughout this project I plan to make use of UML. UML provides many different ways to represent and model the various parts of the system. Use case diagrams for example provide very useful information and help in the elicitation of requirements and also in the building of GUIs. Sequence diagrams also help in the designing of algorithms.

I am going to investigate the use of the Rational Unified Process using Rational's Requisite Pro. I think this could be very useful because UML isn't a complete methodology for development, it is only a modelling language. Rational Unified Process is a complete methodology and could help a lot in the development of this project.

I have a plan to help me in the development of the algorithms and equations needed to implement the simulations. I am going to build a tester application that displays a 3d world in flat 2d from 3 different angles – top, front and side – this will allow me to trace the flight of the ball and see the effects that various factors have on the flight. It can also be used in the building of the various courses. When I get around to developing the 3d engine, I will also build in a 4th view into the tester – the 3d view. This will enable me to make sure that the 3d engine is producing a realistic representation and that it ties in with the other 3 views. Figure 1.4.1 shows my idea for a tester application.

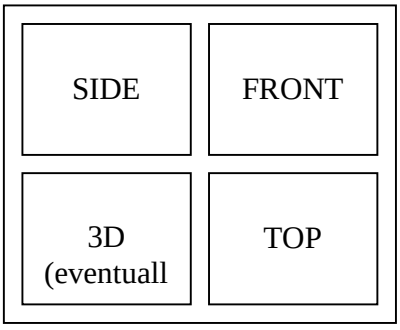


Figure 1.4.1. Rough idea for 3D tester

I plan to implement the program in java. This will mean that it will be playable on almost any platform and, if I make it as an applet, can also be played from the web. The fact that java is fully object oriented means that the design can be object oriented too.

1.5 Project Plan

My plan for the project so far is as follows:

1 Requirements

Analyse the situation. Extract Use Cases and Scenario's. Draw up rough scripts and storyboards of selected processes. Use the data I have found and the processes that I have uncovered to formulate the requirements. Validate the requirements.

2 Risk analysis

Analyse the requirements to see what risks there are to the development process and prepare workarounds, suggest preventative measures and prepare altrenative plans.

3 Design

Complete a system design based upon the validated requirements that I have produced.

4 Implementation & Testing

Implement the design in java, testing as I go.

5 Deployment

Get several different people to use the game and gather feedback.

1.6 Resource Requirements

As this project is going to be implemented in java, there is no particular type of machine or operating system that is required. I have a home computer (Pentium 2 350Mhz, 128MB RAM, 17GB HD) which I can use if I wish, I also have a laptop (Pentium 3 700Mhz, 128MB RAM, 12GB HD) which I can use both to develop on and to demonstrate on.

1.7 Keywords

3d engine, modelling and representation, aspects of realism, golf, game, projectile, simulation.

Appendix B: User Guide

1. Running the simulation

There are several ways to run this simulation. It can be run as a java application or as a java applet.

To run it as a java application, at a command prompt, navigate to the directory containing the .class files and type:

```
java Golf
```

To run it as a java applet, you can either:

a) Open the index.html file, preferably in Internet Explorer but Netscape Navigator will work too, this will open and run the game.

b) At a command prompt, navigate to the Golf directory and type "appletviewer index.html"

c) Go to http://www.madsoftware.co.uk/ and play online.

2. Playing the simulation

The controls are relatively simple. The toolbar to the right hand side allows you to customise every aspect of the shot. To change club, click the next and previous club buttons (See fig 2.1).

Figure 2.1. The club selection buttons.

To change the power of the shot, you simply adjust the Power scrollbar as shown in fig 2.2.

Figure 2.2. The power selector.

To adjust the spin that you place on your shot, you use the two Spin scrollbars shown in fig 2.3.

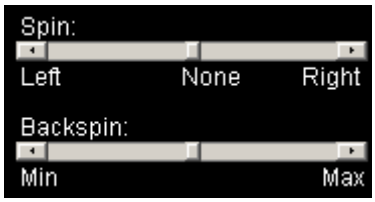


Figure 2.3. The spin selector.

To adjust your aim, click on the screen (see fig 2.4). If you click to the left of the ball, you will look left, if you click right, you will look right. The further away from the ball you click, the greater the angle changes. So, to make minor alterations to your aim, click very close to the ball.

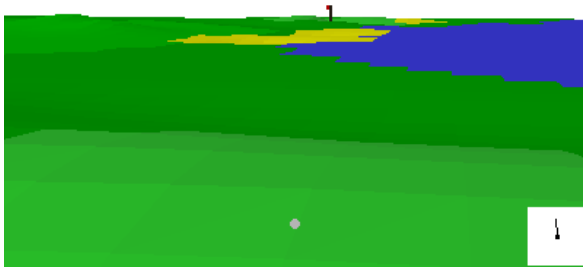


Figure 2.4. The shot screen, click on it to change your aim.

At the bottom right of fig 2.4 is the wind indicator. The direction of the line shows the direction of the wind (up means forward etc) while the length of the line shows the strength of the wind. The wind will continue to change throughout the flight of the ball.

To take your shot click on the “Take shot” button. To restart from the teeing position, click on the “Restart” button. To view the layout of the course and see where you are on it, click the “Overhead View” button. All these buttons are shown in fig 2.5.

Figure 2.5. The control buttons.

3. Features of the simulation

This simulation takes into account factors such as wind and spin, and also factors such as deviations due to the texture of the ground. This means that when you are in the sand, hitting the ball with a driver may produce a very unexpected result.

The green has been designed to be very difficult to putt on. This is deliberate to make sure that the accurate modelling of the rolling of the ball is seen by the player.

Appendix C: Source Code

The source code has been printed in alphabetical order of class names with a header on each page to show which class it is.

This is java source code. It is available from me on request, if you are interested, email me at winchester@imtoosexy.com.